Fraunhofer USA

# SpartanStat

12.6.2017

—

## Team 11

Facilitated by Dr. Qi Fan
Sponsored by Dr. Cory Rusinek

Xingchen Xiao
Nathan Bagnall
Haitai Ng
Yuzhou Wu
Justin Opperman
Brandon Gevaert

MICHIGAN STATE UNIVERSITY

College of Engineering

# Executive Summary

Potentiostats are electrochemical measurement tools that can be used to measure and detect the levels of heavy metals in solutions. Present day potentiostats are large, heavy, and expensive, often costing thousands of dollars. The goal of this project was to create a cheap, portable, and accurate potentiostat that could be used to collect data in the field, or be used as part of a larger unit to analyze blood samples. The potentiostat was able to generate square wave and linear sweep voltammetry, measure currents in the single digit microamp range, and graphically represent the collected data. Validation and performance evaluations were conducted by comparing data from industry standard potentiostat devices to data generated by the developed portable potentiostat.

# Acknowledgement

This work would not have been possible without the support of the Michigan State University College of Electrical and Computer Engineering, and Fraunhofer USA. The design team would like to express their gratitude to our sponsor, Dr. Rusinek, and our facilitator, Dr. Fan, for their patience, guidance, and valuable and constructive critiques of this project. The design team would also like to thank Dr. Albrecht and Dr. Udpa for their professional advice and for keeping our progress on schedule.

The design team would also like to thank the teaching assistants, technicians of the laboratory of the senior capstone laboratory and makerspace for letting us use their technical resources that were required to complete this task.

# Table of Contents

# Chapter 1: Introduction and Background

## Introduction

Fraunhofer USA Center for Coatings and Diamond Technologies (CCD) is a research institute with a primary interest in coatings and diamond technology. This technology is used for a variety of applications ranging from semiconductor development to medical devices to environmental sensors. Fraunhofer CCD has sponsored this project for the express interest of environmental monitoring of heavy metals in aqueous solutions. The primary element that the Fraunhofer Institute is focusing on detecting is lead. Current industry standard methods for analyzing heavy metals in aqueous solutions require very large, immobile, and expensive lab equipment. Fraunhofer CCD is in need of a portable and low cost electronic system that can be combined with their proprietary diamond based sensors in order to collect data away from their lab.

## Background

Before the problem that the team was trying to solve can be understood, a brief survey in electrochemistry is needed. This background contains a description of electrochemistry, voltammetry, and what a potentiostat is used for.

## Electrochemistry

Electrochemistry is the field of science that studies the relationship between electricity and chemical changes. Chemical reactions that involve the exchange of electrons as either reagents or products are called oxidation-reduction reactions (redox reactions). By applying a voltage to the material that is being studied, the rate of a redox reaction can be affected. A standard oxidation potential is the voltage at which a material is most likely to undergo a redox reaction and either gain or lose electrons.

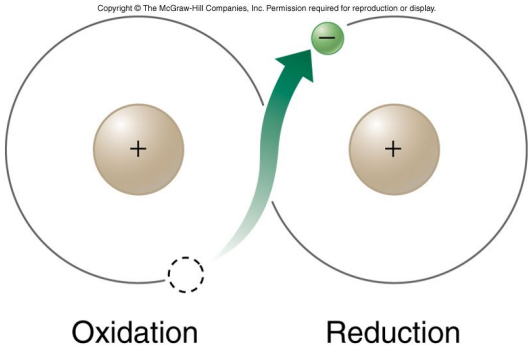


**Figure 1: Oxidation and Reduction**

## Voltammetry

Voltammetry is an electrochemical method that studies how an analyte reacts to an applied voltage. By varying the voltage applied to an analyte, information about the analyte can be gained as the analyte undergoes a redox reaction and generates a current in response to the applied voltage. This can be used to detect the levels of dissolved solids in a solution, as different dissolved materials will generate detectable currents at their standard oxidation potential.

The most common voltammetry method uses a three electrode cell consisting of a working electrode, reference electrode, and counter electrode. The working electrode (WE) is where a controlled voltage is being applied, and is where the redox reaction of interest is actually occurring. The reference electrode (RE) is an electrode which has a very stable and predictable electrochemical potential. The voltage at the RE is measured and compared to the voltage at the WE to get a true sense of what the voltage at the WE actually is in comparison to the analyte. The counter electrode (CE) provides a path for electrons that have been generated by the redox reaction to flow. When the voltage applied at the WE nears the standard oxidation potential of a dissolved solid, the dissolved solid will undergo a redox reaction and generate a current at the CE.

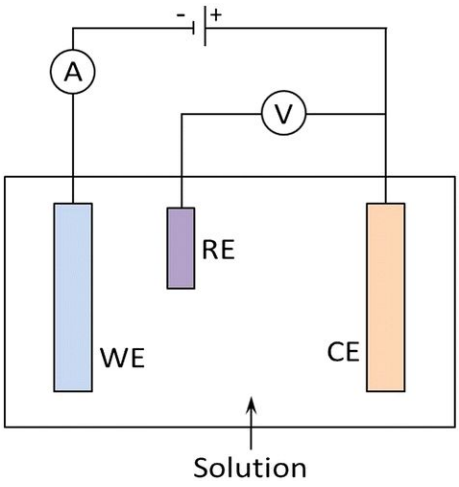


**Figure 2: Electrochemical Cell**

This current generated at the CE will be proportional to the amount of the dissolved solid present in the solution. Thus, by sweeping the potential at the WE across a range of voltages, many dissolved solids within the solution can be detected, and their concentrations can be determined.

The types of voltammetry that Fraunhofer is most interested in are called linear sweep and square wave voltammetry. In these methods, the potential at the WE is varied with time,

and the current at the CE is measured in response. This voltage vs current data is then plotted in a voltammogram.

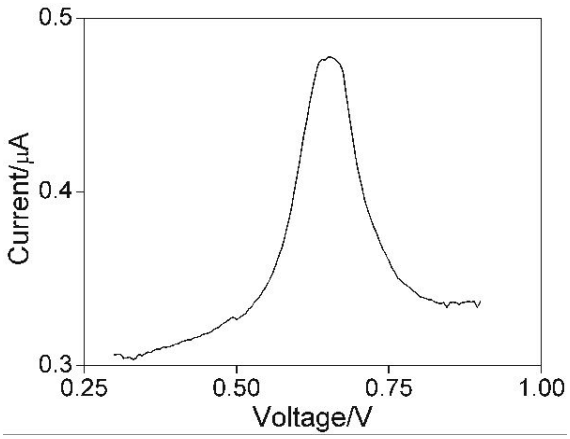


Figure 3: Sample Square Wave Voltammogram

A linear sweep is just a simple increase of voltage where the slope is held constant to see the change in current in relation to a constant increase in voltage. It gives a base level of information about the analyte without requiring massive changes in parameters, a complex setup, or time delays to complete the detection. In square wave voltammetry, a square waveform is generated. Square wave voltammetry accounts for the capacitance of the electrodes by holding the voltage at the WE constant for a length of time. This allows time for the dissipation of charging currents that are caused by the sudden increase in voltage at the WE. This results in very accurate measurements of very small levels of dissolved solids.

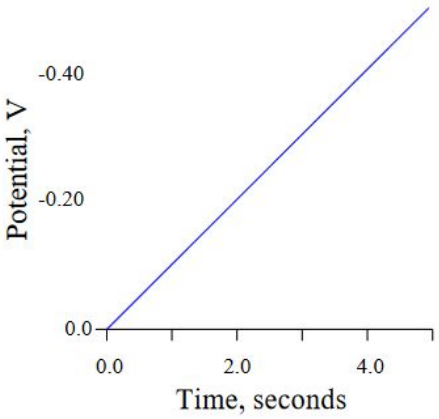


Figure 4: Linear Sweep

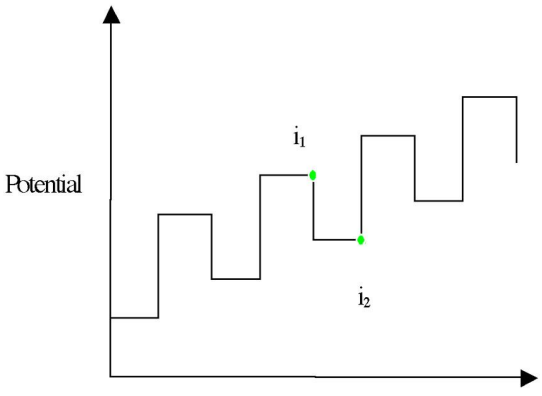


Figure 5: Square Wave Sweep

## Potentiostats

Potentiostats are electronic devices that can control the voltage at the WE and measure back the current at the CE. When connected to a three electrode system, potentiostats can be used to perform voltammetry experiments. The parameters of a waveform can be set by a trained operator and used to collect data on the analyte being measured. Present day potentiostats are typically very large and very expensive instruments, costing thousands or even tens of thousands of dollars. The size, weight, and cost of these present day instruments contains their usage to the confines of chemistry labs.

Portable potentiostats are desired that could be carried by a trained chemist who could take measurements out in the field, or even go “door-to-door” taking water quality samples. Due to the prominence and national attention generated by the Flint Water Crisis, there is renewed interest in the scientific community to find a portable lead sensor that could implement a potentiostat as part of its measurement system. Furthermore, a small potentiostat could be used as a subsystem within a larger blood measurement system that could be used in hospitals to take blood samples and determine the levels of lead within the sample.

## Approach and Significance

The design team plans on developing a portable potentiostat by using cost efficient hardware, a widely accessible standard microcontroller, and commercially available free software. The development efforts of this project will be divided into three distinct categories: hardware, embedded software, and software. What separates our development efforts in comparison to other open source portable potentiostat efforts is our specifications and features that will be incorporated into our potentiostat. Most portable potentiostats do not feature square wave sweeps and do not have the configurable capabilities that will be supported in our design.

The final design that will be implemented will be based on current open source portable potentiostat devices. It is speculated that by referring to these published portable potentiostats, slight modifications will only need to be applied such that it can meet the specifications defined by our sponsor.

Due to increasing levels of pollution and higher levels of contaminants in natural resources, portable sensing of unwanted compounds in liquid based solutions is a growing industry. The significance of this project is it could aid in the future development of portable potentiostats.

# Chapter 2: Exploring the Solution Space and Selecting an Approach

## Critical Customer Requirements

The objective of this project was to build a portable and low cost potentiostat with the following operational functionalities and specifications:

- Must support square wave and linear sweep voltammetry
    - Minimum output voltage step size: 10mV
    - Frequency of output voltage waveform: 15 Hz to 25 Hz
    - Output voltage range: -2V to 2V
    - Average time to complete one cycle: 25 seconds
    - Minimum current sensitivity: 1 microAmp
- Data must be outputted to an external platform (*laptop,* phone)
    - Capable of presenting data graphically as a voltammogram
    - Capable of presenting data in a human readable form as a text file
- Must be handheld and portable
- Must be powered using an external power supply

## Research Process

Before prototyping could be started, research needed to be done to understand more about how potentiostats function, and about the potentiostats that are currently available on the market. The team looked at existing portable potentiostat designs, how they functioned, and the areas in which they were lacking. Referenced potentiostats included the CheapStat and the DStat.

The team also found a few research papers from universities that contained existing designs, including a research paper from the Universidade de São Paulo. Some of these designs were used as starting points for our initial hardware designs, although much of these designs ended up being changed.

Referring to current state of the art in portable potentiostat technology helped to give us a foundation for our hardware designs, and estimates of target cost and performance.

## High Level Architecture and System Planning

It was decided that the design would be split up into 3 main parts:

- Laptop Application
- Embedded System
- External Hardware

The laptop application was to function as the graphical user interface (GUI). This would be used to issue commands to the potentiostat, graph data, and analyze data. The embedded system was to function as the unit that would drive the external hardware, and be capable of capturing raw data from the external hardware and transmitting it back to the laptop application. The external hardware would be used to convert digital signals from the embedded system into analog voltage waveforms, and convert analog current readings into digital signals that could be read by the embedded system.

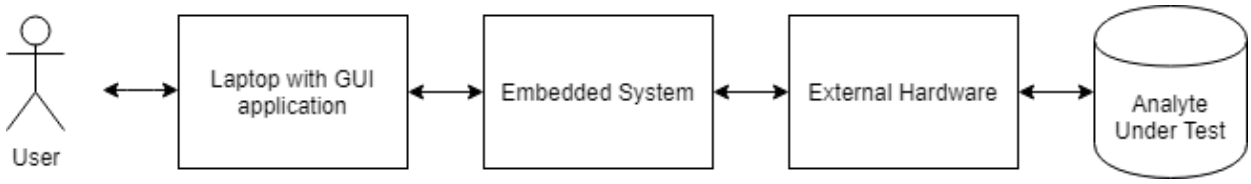


**Figure 6: Architectural Diagram**

A FAST diagram was also created to help organize the flow of the project.

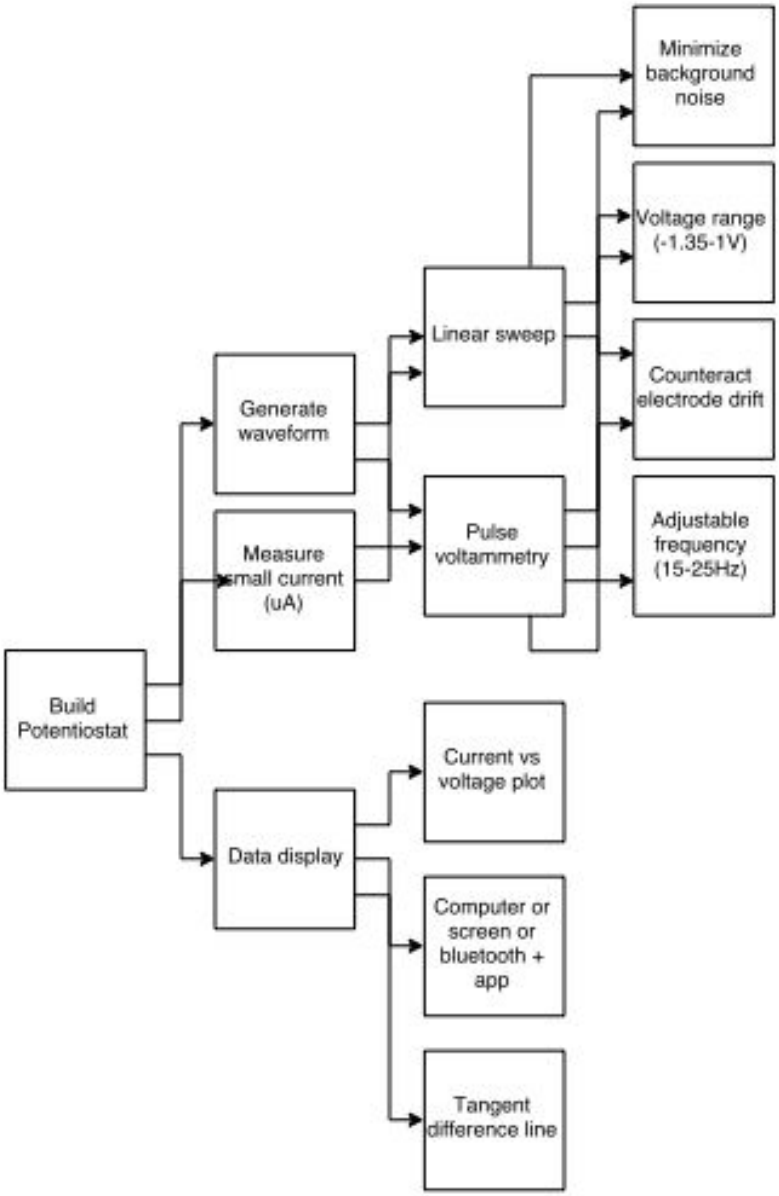


Figure 7 : FAST Diagram

## Initial Budget

During planning, we came up with a projected project cost outline as follows:

# Project Cost

| | Subgroup Descriptions | | Budgeted Cost | | Actual Cost |
|---|---|---|---|---|---|
| PROJECT DEVELOPMENT | Purchase Microcontroller | $ | 36.12 | $ | 36.12 |
| | Op Amps | $ | 25.00 | | |
| | Power Supply Circuitry | $ | 30.00 | | |
| | Misc | $ | 50.00 | | |
| | **Subtotal** | **$** | **141.12** | **$** | **36.12** |
| FINAL DESIGN | PCB | $ | 80.00 | | |
| | Enclosure | $ | 10.00 | | |
| | Power Supply | $ | 20.00 | | |
| | Additional Circuitry | $ | 50.00 | | |
| | Misc | $ | 50.00 | | |
| | **Subtotal** | **$** | **210.00** | **$** | - |
| OTHER COST | Other cost | $ | 50.00 | $ | - |
| | Other cost | $ | - | $ | - |
| | Other cost | $ | - | $ | - |
| | **Subtotal** | **$** | **50.00** | **$** | - |
| **Subtotals** | | **$** | **401.12** | **$** | **36.12** |
| Risk (Contingency) | | | | | |
| **Total (Scheduled)** | | **$** | **401.12** | **$** | **36.12** |

**Figure 8: Projected project budget**

This detailed all of the planned costs for our budget without using the total in order to allow for more purchases if necessary.

## Initial Schedule

Our initial schedule that we intended to follow is outlined below in a minimalistic overview. It outlines the primary sections necessary to complete the project. Like in our budget, we gave ourselves room to change the schedule if need be by not pushing validation and documentation up until the last minute.This can also be seen in the full size Gantt chart in appendix III.

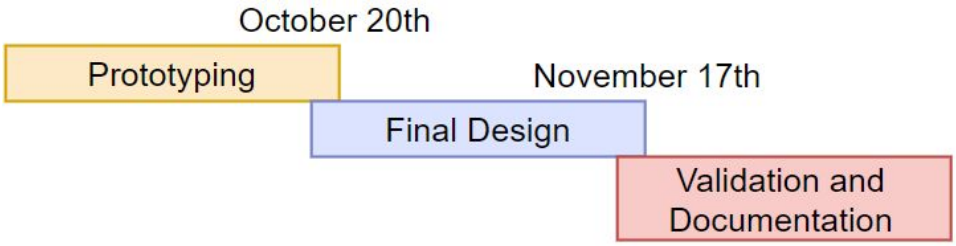


**Figure 9: Minimal Gantt Chart of Initial Schedule**

# Chapter 3: Technical Description of Work Performed

## Hardware Design: Overview

For our potentiostat to work properly, we needed to be able to drive a voltage at the working electrode, and read back current generated at the counter electrode. This allowed us to break down the hardware design into three parts:

- Voltage driving circuit
- Current reading circuit
- Power supply circuit

## Hardware Design: Voltage Driver

The first iteration of our voltage driving circuit design was based on a research paper published by the Universidade de São Paulo.

The design in this paper was built to output a voltage range between 1V and -1V. However, our design needed to cover a larger voltage range between 2.0V and -2.0V. The resistor values in this design were altered to suit the larger voltage range. The op-amps we used were LF411s as they had very small input bias currents and small input offset voltages, as well as a slew rate significantly faster than the period of the square wave that we were trying to create. In addition, all supply voltages were replaced with ±5V, in order to reduce the complexity of the power supplies needed.

The first iteration of our hardware design used pulse width modulation (PWM) in order to convert a digital signal from the embedded software platform into an analog signal. The microcontroller that we were using was controlled using software to output a PWM signal from one of its pins, and allowed for the creation of 256 different output values (2^8 values). The PWM signal was put through a first order low pass filter in order to convert it to a constant voltage between 0 and 5V. However, after testing out this design, it was discovered that the PWM signal was very noisy, most likely due to the switching noise caused from switching an output pin on our microcontroller on and off very quickly. The noise was sometimes as high as 1V to 2V.

In order to reduce this noise, the first order low-pass filter was replaced with a fourth order low-pass filter. This did significantly reduce the switching noise, but still caused many voltage spikes that sometimes reached about 500 mV. It was decided that a solution had to be found that didn't use PWM as its main digital-to-analog conversion mechanism.

The solution to the problem was to buy a dedicated digital-to-analog converter (DAC) that used a non switching mechanism to do the conversion. When deciding on a DAC to purchase, resolution was considered to be the most important factor that would impact performance. Our design requirements dictated that our output voltage to the working electrode must be able to swing between -2V and 2V, and have a minimum step size of

10mV. A 5V range with 10mV per step meant that we would need at least 500 discrete output points.

However, it was decided to purchase a DAC with a step size an order of magnitude higher than the minimum resolution needed in order to allow flexibility in the rest of the design, and to provide better performance. A 12-bit DAC was purchased that provided 4096 discrete analog values that could be output to the working electrode. The 12-bit DAC was also purchased due to its compatibility with our proposed power supplies, and because it featured a parallel input setup that we thought would be easier to implement and debug.

The DAC datasheet was referenced to construct additional circuitry that got it functioning properly. We were successfully able to verify that the DAC provided a much higher step size resolution of 1mV. One issue that was run into concerned the DAC's WR (write) pin. In order for a new digital value to be written to the DAC, this pin needs to be low. Originally, we just tied this pin to ground, so that any new value at the DAC inputs would cause the DAC to output. However, we realized that the outputs from the Arduino do not all change at the same time. This causes unpredictable behavior at the output of the DAC when transitioning from one state to the next. We fixed this problem by tying the WR pin of the DAC to a pin on the Arduino, and toggling the pin after a short delay when the outputs on the Arduino change. After making this adjustment, we were able to verify that the DAC worked, and provided much more accuracy than the previous method of digital-to-analog conversion.

At this stage, a design flaw was noticed that was resulting in an output voltage range that was much less than the full 5V range that we were designing for. It was determined that one of the op-amps that was being used was clipping the output signal. After some testing, it was determined that the drop across the op-amps was about .5V for the positive supply, and 1.5V on the negative supply. It was then decided that the power supplies needed to be altered in order to provide ±6V in order to give the DAC output the full voltage range it needed. The measured output from the DAC was able to vary between 0V and 4.823V.

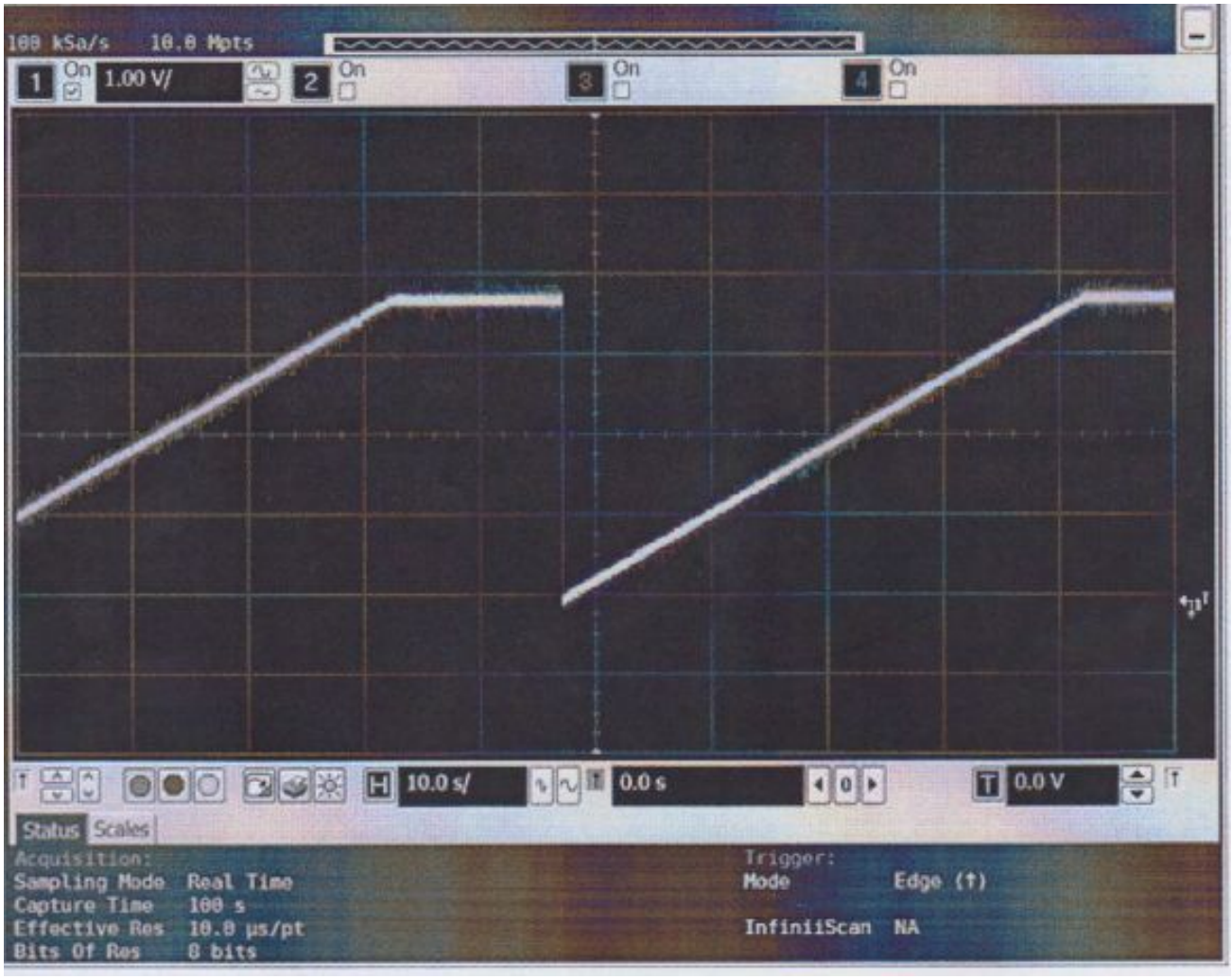


Figure 10: Output of DAC with Clipping at Top with ±5V Supplies

However, the output to the working electrode had to vary between -2V and 2V, so a circuit was needed that could shift the DAC output voltage down by about 2.4V. The circuit that was used for this was an inverting op-amp summing circuit. Two design iterations were tested. The first design iteration output values between -2.0V and 2.8V. It was decided to try and shift this voltage down a further .4V in order to center the range properly. The second iteration was able to output a voltage between -2.48 and +2.34V. This was adequate to fulfill our design range requirements.

The final part of the voltage driver circuit was a circuit that would ensure that the control voltage signal output to the working electrode was with respect to the reference electrode. This was accomplished by using an op-amp circuit that takes the control signal as the non-inverting input, and the reference electrode as the inverting input. This drives the voltage at the working electrode with respect to the reference electrode to the voltage of the control signal with respect to ground.

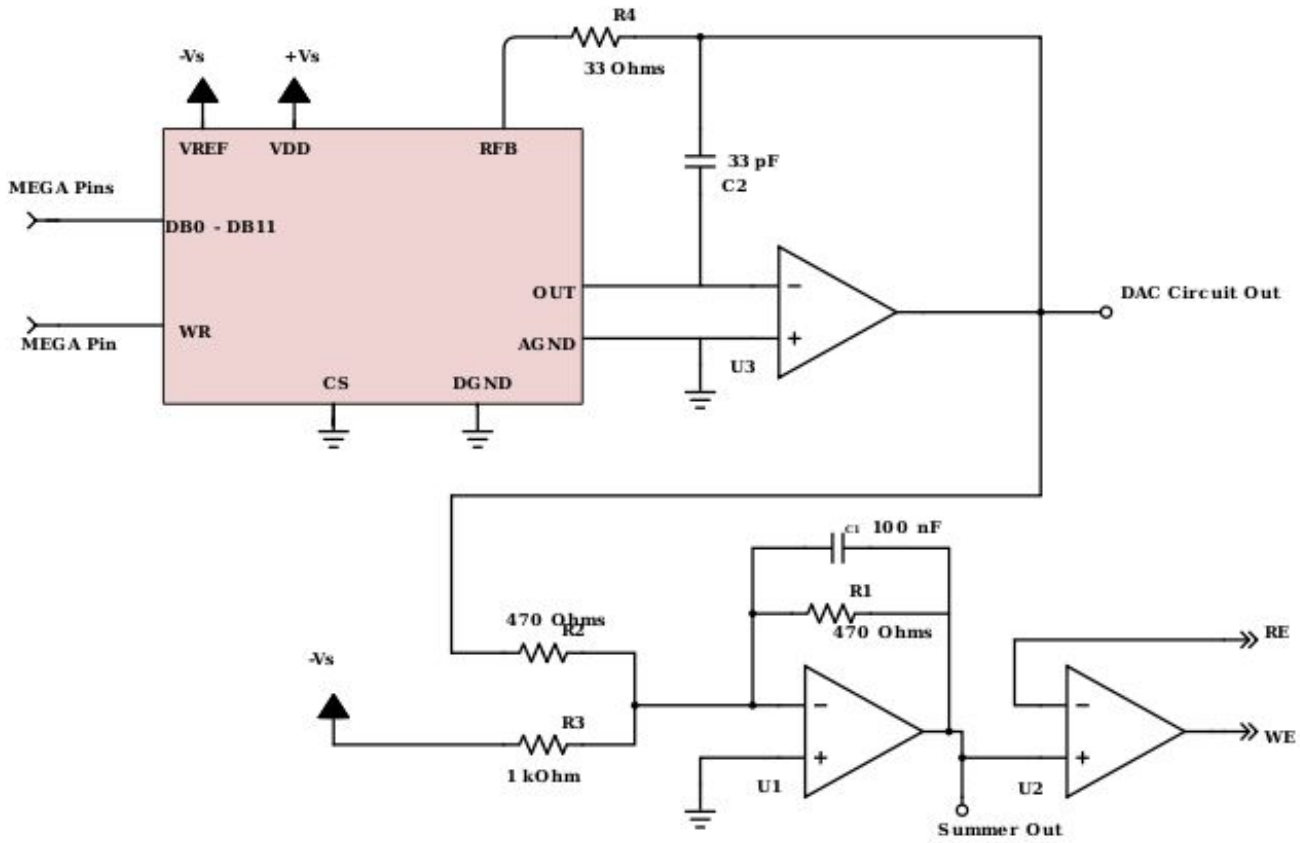


Figure 11: Final Voltage Driver Circuit

Our final voltage driver circuit can be seen in figure 11 with supply voltages of ±6V. The pink block is the MX7545LN DAC.

## Hardware Design: Current Reading

The first iteration of our current sensing circuitry used the Arduino's built in ADC to read a voltage at the input. In order to convert a current from the counter electrode into a voltage, a transimpedance amplifier was used. The transimpedance amplifier would output a voltage 27000 times the current received, meaning that 1uA would translate to 27mV. At 100uA, we would get a voltage of 2.7V. At -100uA, we would get -2.7V.

However, the Arduino ADC was only capable of reading positive voltages between 0 and 5V. So, a level shifter was built that would sum a voltage of about 2.7V to the signal coming out of the transimpedance amplifier. After testing this unit, it was would that a current of -100uA gave 5.05V at the ADC, and +100uA gave -.35V at the ADC. This satisfied the total range of current that we were required to be able to read.

Overvoltage protection using zener diodes was also implemented at the ADC input pin, in order to prevent the voltage read at the ADC from being less than 0V or greater than 5V. Any voltage outside this range could damage the ADC. Implementing the zener diodes decreased the effective range that the current could be read to ±85uA.

However, a few problems with this design were discovered. The built in ADC on the Arduino has only a 10 bit precision, which was significantly reducing the accuracy of the current reading circuit. In addition, the current reading circuit was very noisy. As the current reading circuit had to be capable of reading current in the single digit microamp range, even small amounts of noise would cause inaccurate readings. The team decided to look into methods of reading current that would be less noisy, and also decided to use a better ADC in order to increase range and accuracy.

In the development of the actual current sensing aspect of the final device, it was desired to focus massively on keeping the accuracy and precision high while keeping the cost and the size of the device reasonable. This was completed by looking into the present day methods for how current sensing is being completed on multiple devices and in laboratory settings. Overall, there was a comparison of these methods based on size, cost, accuracy, and precision in order to determine which method would be the best to avoid problems in the design and to keep the overall parameters within our requirements.

The methods that were researched all varied in how they sensed the current. The first was using shunt resistors or Rogowski Coils to measure the current directly. The method of using shunt resistors is very common given the ease of use, the high accuracy, and the fact that this method allows quick swaps if problems arise. The overall issue with this design though is that it is limited by the power dissipated by the resistor. The Rogowski coils also face similar issues given that they are also connected directly to the circuit, but they are only used for AC current, which does not always apply with current sensing in Potentiostats as DC voltages are being measured and AC results are not likely. The other methods such as Current Transformers, Magnetoresistive sensors, Hall Effect Sensors, and Fluxgate sensors all use the method of indirect current sensing by measuring the magnetic field of a wire that passes near a section of the device. These methods vary in their capability, use, and overall accuracy, so it was decided not to use these methods for various reasons. The Current Transformers were not used because they are primarily used for AC supplies where high current is measured such as in power grids. Since we are testing for very low current, this method would not be as useful without the development of the actual components. Magnetoresistive sensors were eliminated from our design choice because we wanted to have high accuracy which is not usually possible for Magnetoresistive sensors without a high cost increase for a marginal accuracy increase. Hall Effect sensors were eliminated because they are inherently very noisy. This would remove the accuracy that we want as the noise variation would be higher than the accuracy requested by the project sponsor. The fluxgate sensors were removed because they are complex, require very low mechanical tolerances, and the overall size of the components are much larger

and heavy in weight to get the accuracy that we required which would not meet the requirements set by the project sponsor since it would not be as portable given the desired accuracy. However, if the cost of these types of sensors were to reduce massively in size, weight, and general cost, they would be a great future addition because they have the highest level of sensitivity of all of the current methods, lowest noise of the current methods, high accuracy, low level leakage current, and the temperature drift is very low. As a result, it was decided that we should base our current sensing around the use of a shunt resistor.

The next step in development of the current sensing section of our device was to select an appropriate resistor. The selection of this resistor is done to keep the voltage high enough to measure given the current limit of the original device while keeping the resistor high enough in value to reduce the overall noise. The shunt resistor is selected based on the desired range, the accuracy required, the deviance from the expected value, and the resistance value of what is being tested. The range is very important overall as it limits the final resolution of the output and the necessary accuracy of the resistor value. The deviance from what the expected value should be given an applied current is important since the overall resistance could change as much as 5 to 10% error. For our design, we decided to go with the standard 2 connection resistors, since the 4 connection resistors which reduces this deviance would not be necessary since we are not measuring in the picoamp or nanoamp range. The resistance of what is being tested and the accuracy play much more major factors since the shunt makes the resistance of the measured sample act as if it is connected in parallel. For instance, from the average of the sample data supplied by the sponsor, it was calculated that there was an effective resistance of at a minimum 220k Ohms. Combined in parallel with an example 1k ohm shunt resistor, the effective combined resistance would be 995.475 ohms. This would cause a decrease of 0.45 % in the current reading. This increases much more as the value of the shunt resistor increases.

After selecting the right resistor, the next step in the development of the current sensing section was to increase the output voltage of the resultant shunt to a usable level for sensing. Given the fact that the design group chose 10 ohms for the resistor shunt, a rail voltage supply of -6 volts to 6 volts, and the maximum current we expect to be in the range of -1 mA to 1 mA, we selected our gain value for 500. This would let our design group have 1 mA be the point where 5 volts is the output. Also, since 1uA is equal to 10 uV after the shunt resistor and the gain was selected as 500, the effective voltage after the gain aspect of the circuit would be 5 mV. This is a value that most microcontrollers can read as an input. After this was selected as a viable range, the design team then designed an op amp based circuit to set the desired gain. This op amp circuit was set up using a non-inverting op amp as the design team felt that keeping the sign of the voltage equivalent to the sign of the current would be less confusing to the groups coding the microcontroller. The design team felt that a 499k resistor and a 1k resistor would be the best method to get the required gain since these resistors would be reasonable in size, accurate enough given

their cost, and yet not small enough to allow drift to be a major factor. However, the selection of the op amp in this section is more important overall for the accuracy and range. The op amp sets the effective bias current, the maximum gain at different frequencies, the difference across the positive and negative terminals, and the allowable range of the output. For this project, the design group used the LF411CN op amps to test the basic real-world functionality of the current sensing. After the functionality was verified, it was decided to switch to the OPA735AID op amp. This op amp has zero drift as the effective difference across the input terminals was 1 uV normally and 5 uV at a maximum. Given our gain, the effective loss of voltage was 500 uV while the output would be 5mV if 1uA was applied. This is equivalent to a .1uA loss in accuracy.

The next step in the development of the current sensing section was to reduce the possible error. This was effectively done with a comparator set up to act so that the error is subtracted away. The design group set the positive input to the actual tested reading while the negative side that is subtracted away is setup to an exact copy of the original circuit used for the current conversion via a shunt resistor and the gain circuit connected to a banana connector. This design choice would allow the sponsor to connect filters or grounds to effectively offset away errors that are detected due to floating grounds or reference drift. The result of this section when not acting to eliminate floating ground or reference drift would remove the 500uV inaccuracy and the result would be a 1 to 5 uV inaccuracy instead if the same op amp is used, which is what the design group decided would be the best method. The circuit set up in a comparator mode can be made into a basic subtractor by setting all of the resistor values equal in the section. However, the selection of these resistor values is very important for the overall design. The resistor values selected here greatly limit the effective range of output voltages as these resistors have an effect on the voltage range and can cause the voltage range to be artificially low. The design team tested the current sense in ADS to verify the resistor values selected as well as the total circuit to this point to verify the total feasibility of this design. The resistance for the comparators all had to be above 400 ohms to allow the current range desired to be possible. The design team at the same time had to limit this resistance overall as the higher the resistors increased, the more the current offset became an issue and thus the accuracy of the device. It was decided to select the resistor value at 470 ohms as this would allow an effective range of -5.030 volts to +5.030 volts.

The results of the simulation are shown below. Figure 12 shows the actual total design with resistor values shown and the simulation parameters. The error correction section had white noise applied to simulate being connected to the air or to a stable ground. Figure 13 shows the input range and the total output as well as the the amplification section output and error correction output. They are shown in order starting at the top left and going clockwise. The first figure of figure 13 is the current applied along with the step in current. The horizontal axis is the actual step in the sequence. The next graph is the output of the system as a result of the input current. The range of -1mA to 1mA is shown as being viable as well as the -5 volt to 5 volt output from the current sensing. The next graph in the

sequence is the error correction. This graph shows that the error increases and works accordingly to reduce error. The final graph is the output from the current to voltage conversion and the amplification. This graph represents what we would expect to be a result if no error was removed.

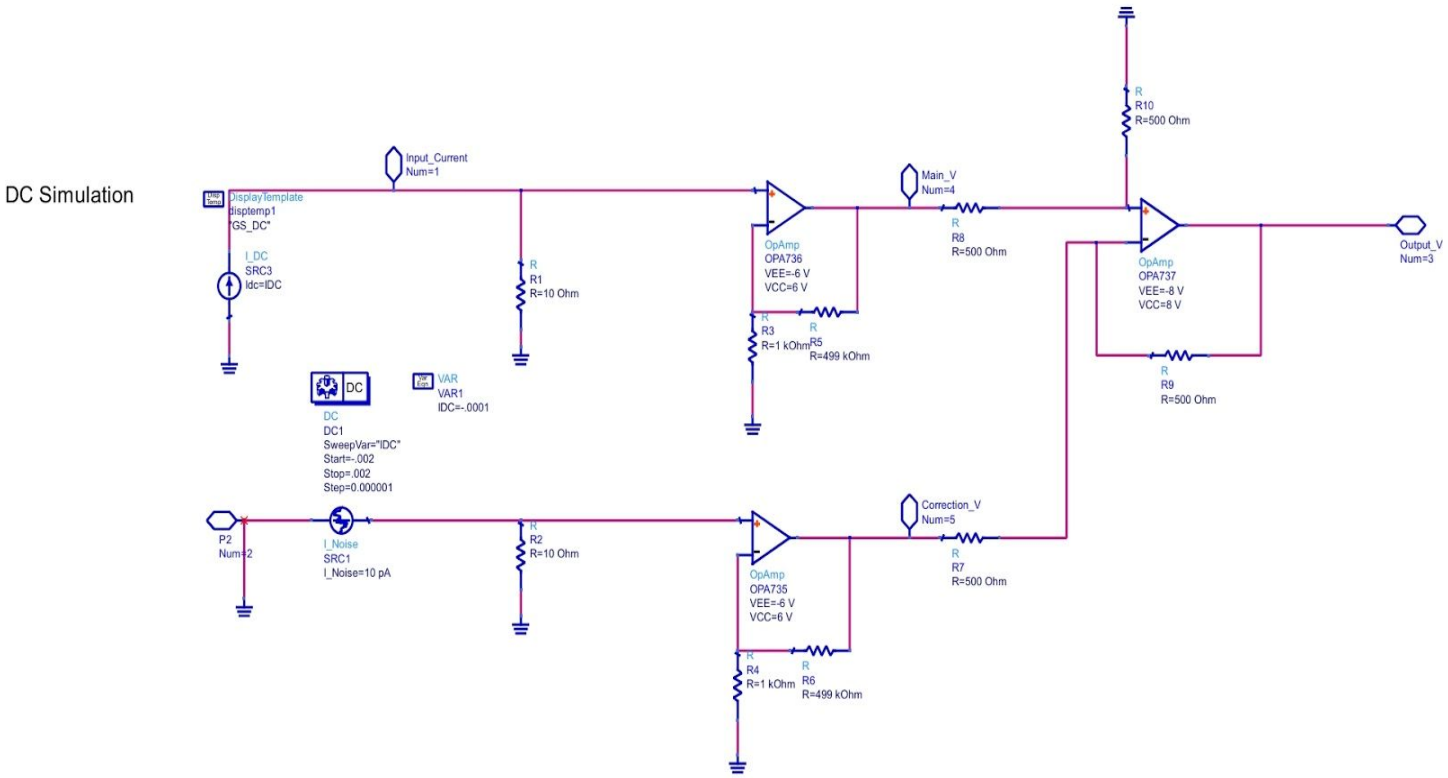


Figure 12 : Final Current Sensing Design (without ADC)

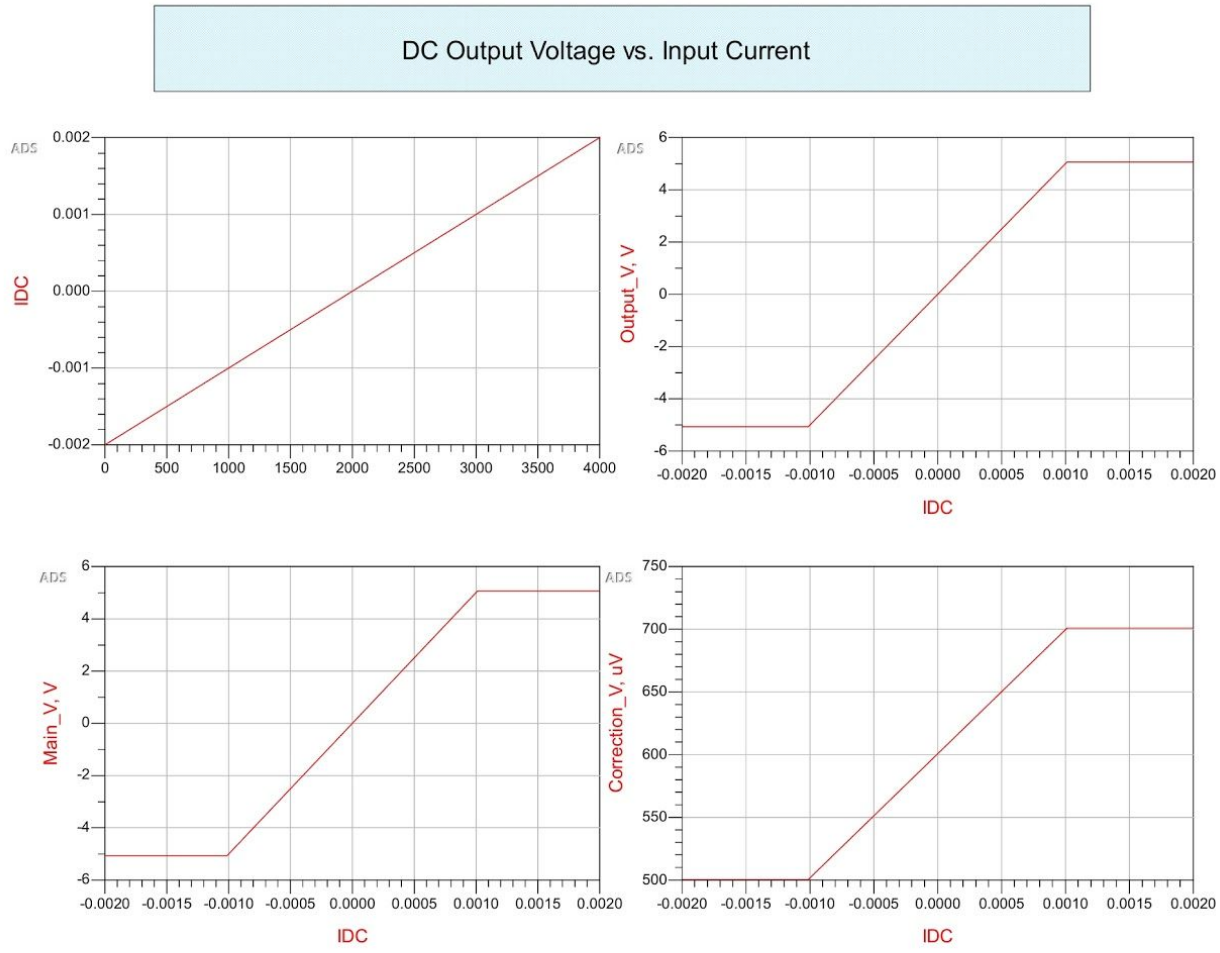


Figure 13: Analysis Graphs from ADS

The last section of the current sensing was to connect it to the ADC. The ADC was chosen effectively based on cost, voltage range, and the number of bits. The external ADC was used to achieve a better accuracy. The accuracy of the Arduino device the design team selected was effectively 5mV. However, the ADC selected from Adafruit would have an accuracy of 3.72mV. This means that for our current sensing the overall current could be inaccurate by 186 nA in either direction. This was deemed to be acceptable as the overall current sensing would only add about 218nA of total inaccuracy if every resistor component selected was at the maximum or minimum possible error in a way to cause the highest amount of error from what should have been detected. This was determined by finding the maximum error of each section. The maximum error from the shunt resistor would add 1.01 times the current, the gain section would add 1.0020 times the voltage, and the subtractor could add at most 1.0201 times the voltage. This circuit was purposely over

designed by the design team to fully minimize loss as the high accuracy was deemed as a very important aspect of the overall device.

## Hardware Design: Power Supply

For our power supply design we had a few considerations. We needed to have a positive and negative rail in order to support the op amps we had chosen, in addition to a very stable voltage to limit noise as much as possible. To make it more portable, we were also interested in supplying power with batteries. We came up with a few designs that met all of these. Two of the preliminary designs are detailed in the figures below:

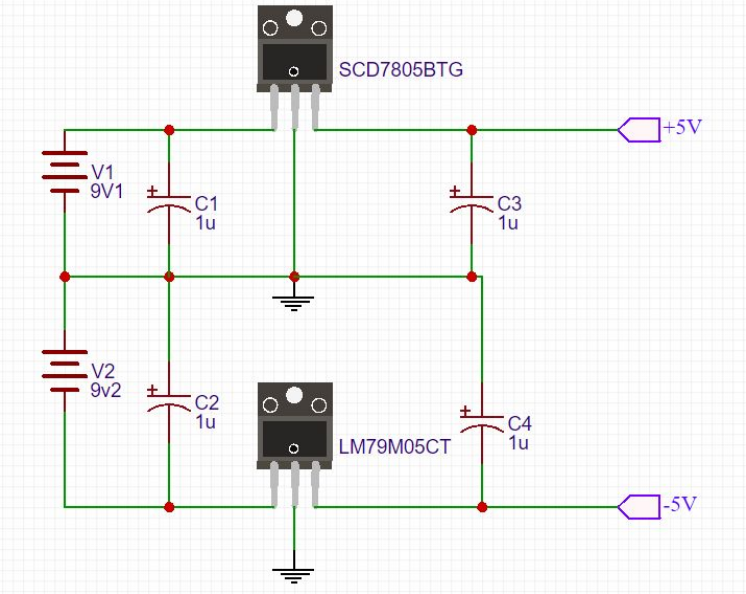


Figure 14: Preliminary Design One

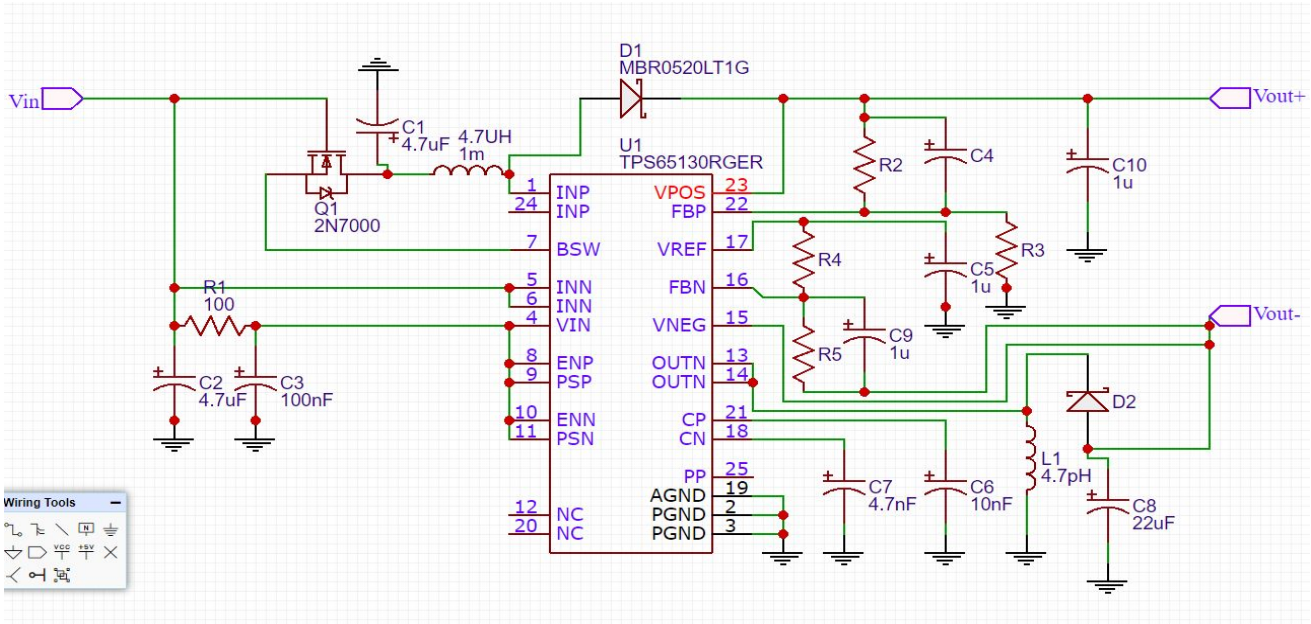


Figure 15: Preliminary Design Two

Although all of these designs met the requirements and output enough power, these first two designs had limitations. The first design uses set point regulators which output a constant +5V and -5V. Although this would have worked in theory, we decided it would be useful to have some adjustment to allow for changes in the power requirements of the design. This worked out in our benefit since we ended up adjusting the required voltages to ±6V which would have made the first design unusable. For the second design, the power supply would have been more complicated, but significantly smaller. Since it used a surface mount IC, it would have been difficult to work with. It allowed for an adjustable output like the final design below, but had a maximum output of 500mA, which is more than enough in theory, but our circuit drew more than this on more than one occasion due to the nature of electrolyzing the electrochemical cell. Additionally, it used switching regulators which would have added additional noise to the system, albeit being more efficient in the process. This leads to the final design, which although not perfect, does match all of the requirements the best. A necessary improvement would be to replace this with a rechargeable system or use batteries with a higher capacity due to the rate at which the batteries are dying which was unexpected due to the seemingly low power usage of our circuit.

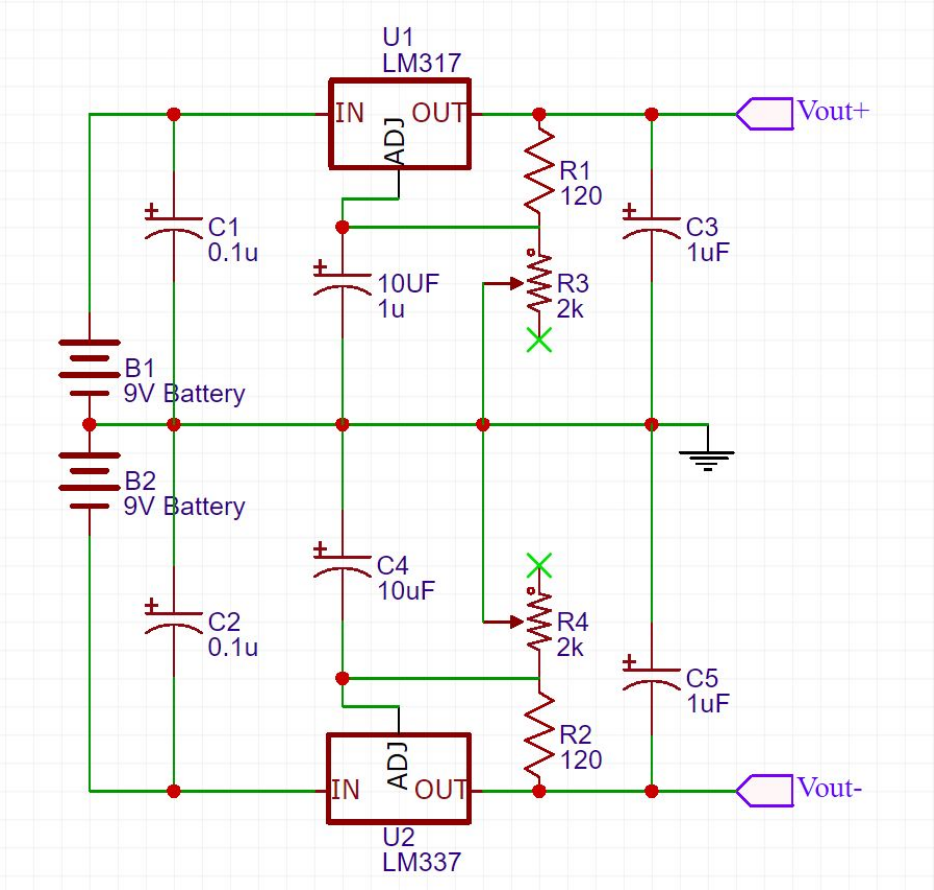


Figure 16: Final Power Supply Schematic

## PCB Design

The development of the PCB was limited by the time we had and the overall limitations of our equipment. The board was developed by the design team in the software available through Easy Eda. This software allowed the design team to develop an effective PCB design that was checked for accuracy. This design is shown below:

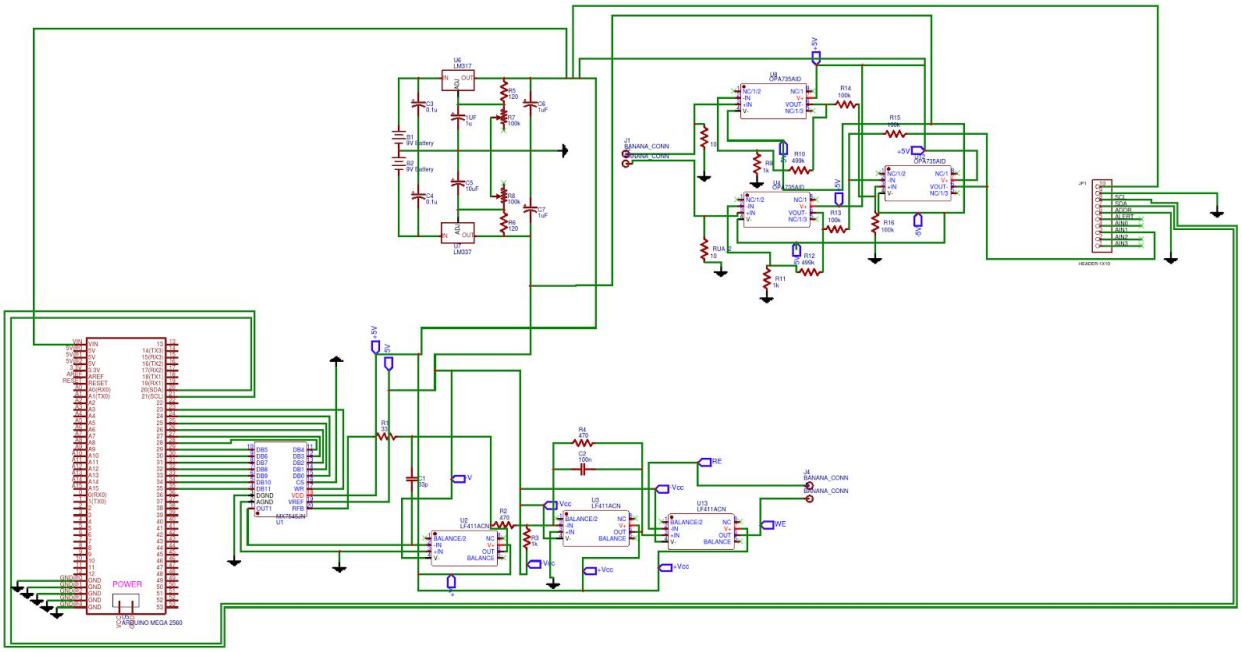

**Figure 17: Final Circuit Design**

While printing the actual PCB, the design team was limited. The design team decided to lump sensor components near each other since the leads from the sensor provided by the sponsor were fairly short. The power supply aspects were moved away from the sensing and output sections to help reduce potential coupling of the magnetic fields at short distances. The DAC was placed close to the pins of the Arduino Mega to keep the connections shorter and minimize errors. The ADC was placed so that it would not block other components as it was its own PCB while also keeping the size of the overall device limited. The other components were all placed to keep the design smaller.

The device size was related to the limitations that the design team did not have access to surface mounted device setups that would help us effectively make the device smaller and have a lower cost. Surface mounted devices often have lower costs for high accuracy compared to through hole connections. However, the use of through hole for most of our components still allowed the design team's design to meet the requirements of the sponsor for size as well as cost. The design team chose the parameters of the PCB by setting minimum track size, minimum distance between tracks, and minimum clearance for all components to 25 mil as these would allow tolerances to help avoid issues. We also

selected the pad size for the via to be 50 mil and the hole size to be 40 mil to also minimize errors. These tolerances would allow the design team to have the board printed by the ECE Shop which was a way to save on costs while also having the delay between when the design team submitted the files to when the PCB was received to be minimized. This did impose its own constraints and potential issues with the overall board as the accuracy of the PCB printers is not up to the same quality as those found in very professional design firms as the manufacturing processes are completely different. The shop does not do through hole connections, so all connections must be remade with soldering wires to connect vias. We faced a few issues with the actual PCB functioning correctly as it was decided to solder all the components and then make the changes necessary to have the board function properly. This method was ineffective at developing a fully working model for our PCB given the possibility for mistakes in both the setup of the board and the production of the board. The overall design of the PCB is included below:

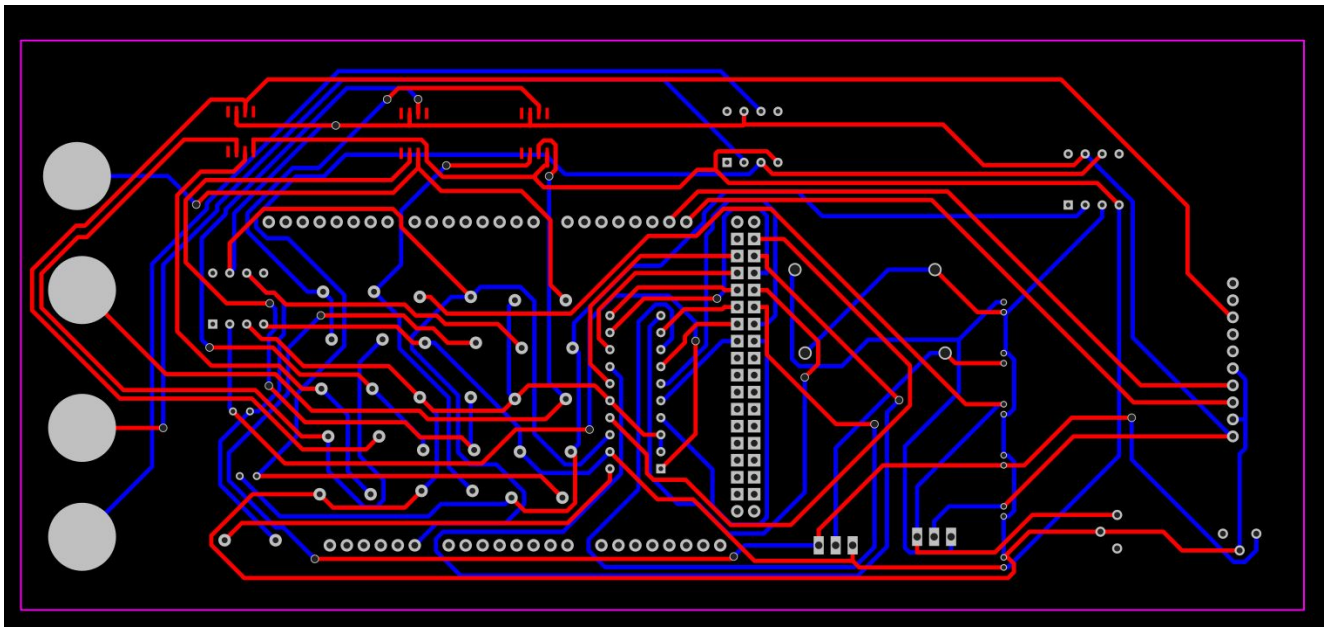

Figure 18: Final PCB Design

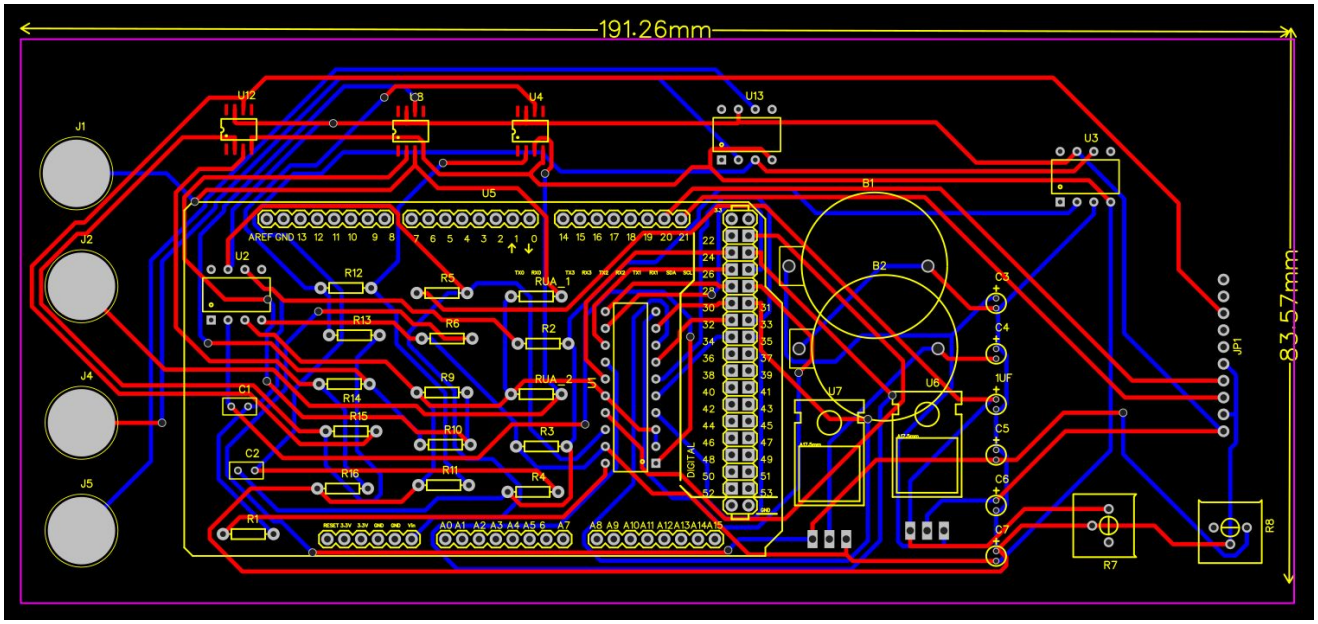


Figure 19: Final PCB Design (Silk Screen and Dimensions)

## Enclosure Design

For the enclosure, the team focussed on function over form. The final design can be seen in the figure below:

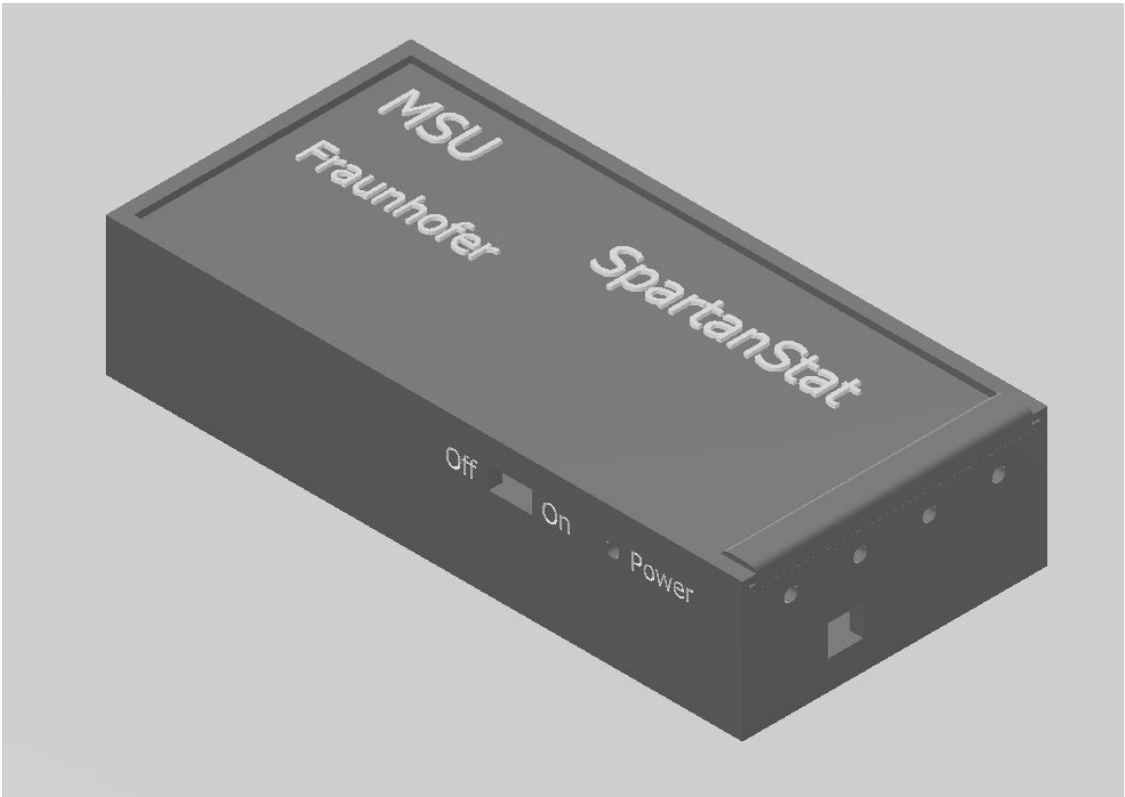


Figure 20: Final Enclosure Design

The design primarily relied on making it small while still allowing room for changes to the PCB size. In the future, this could be made vastly smaller with a single PCB design instead of the current design which attaches to the Arduino Mega. This was done to increase the simplicity of board manufacturing in addition to making it easier to troubleshoot. Due to these constraints, the enclosure's final dimensions are 9" x 4.35" x 1.975." Full annotated drawings can be found in appendix III.

## Embedded Software: Overview

We needed to design an embedded system that could properly drive our hardware, and collect raw data. The embedded system also needed to be capable of communicating with the labVIEW application. These requirements allowed us to divide the embedded system design into multiple parts:

- Microcontroller selection
- Linear sweep generator
- Square wave sweep generator
- Current reader
- Transmission with Laptop

## Embedded Software: Microcontroller Selection

In the beginning, our team planned to use the MSP430G2553. External resource that were referred to during the development include the Texas Instruments webinar series, technical resource manual, and controlSuite. During the first demo the design team was able to configure the GPIO pins to accept a voltage, and plot the voltage in real time using Code Composer Studio's integrated waveform generator. Afterwards the MSP430G2553 supported PWM output.

After conducting further research it was later concluded that this particular model of the MSP430 did not have the internal peripherals that were suitable to meet the specifications of the project. The MSP430G2553 is lacking a higher resolution digital analog converter. The major reason why this model was selected was because this model of the MSP430G2553 is widely available in the ECE Capstone lab, as it was used in the previously required ECE 480 laboratory exercises. The ECE480 Design team then transitioned and focused on learning how to integrate the Cypress microcontroller. Resources that were used to learn how to integrate the Cypress Microcontroller include the PsoC 6 tutorials instructed by Alan Hawse; resource documentation. As stated in our proposal, this device had very limited documentation, thus making development very difficult. In addition it was difficult to sync with other non-cypress products.

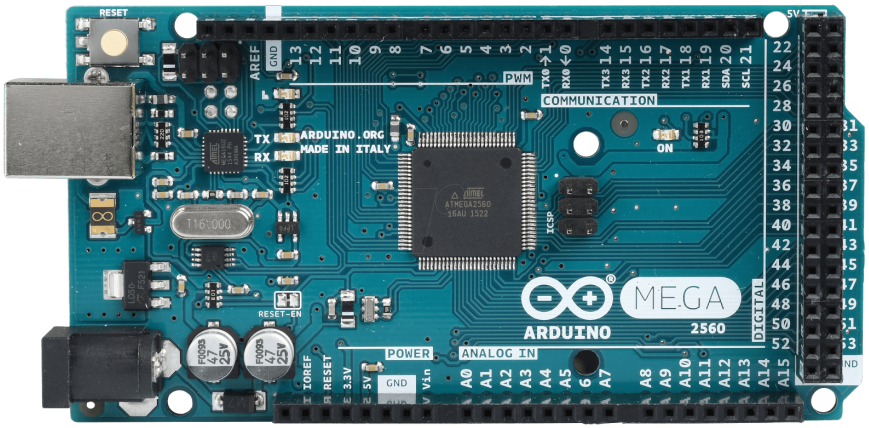


**Figure 21 : Arduino Mega 2560**

After considering the previous options, the team decided to use the Arduino Mega as the integration platform connecting our hardware to our software. This device is able to receive mixed signal inputs (analog versus digital) and is synchronized with our LabVIEW user interface. Data and user input serial commands can be transmitted and received via the USB connection cable. This device was programmed using Arduino IDE, and utilizes embedded C and C++. The Arduino IDE is synchronized with a library developed by the ECE480 Team 11 Team.

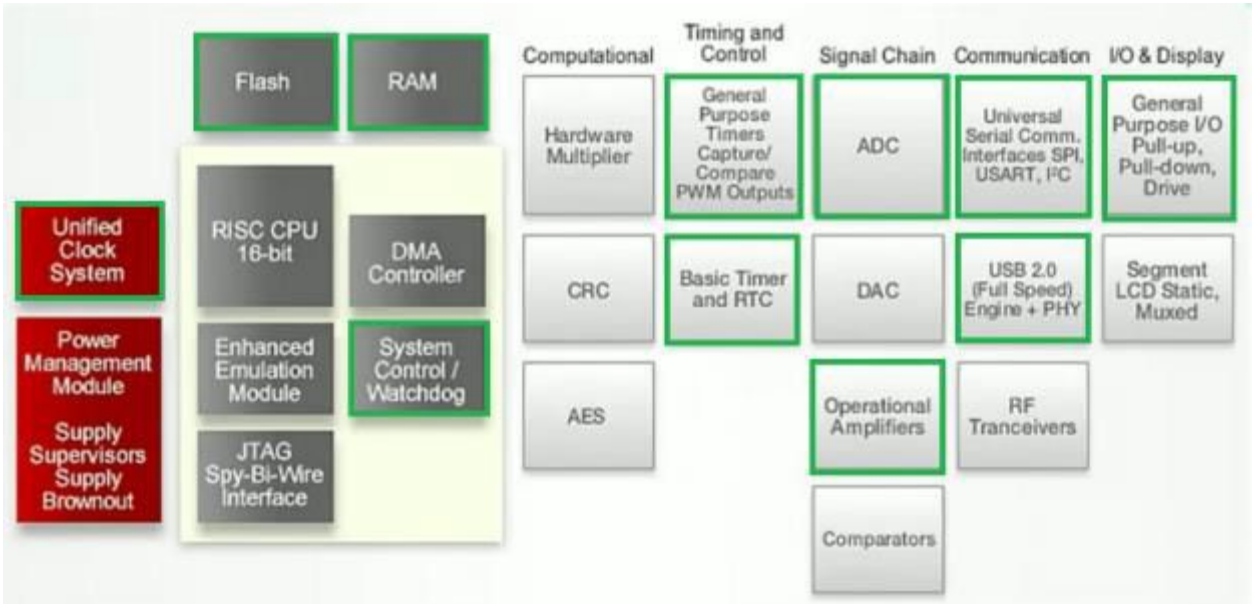


**Figure 22: Arduino Mega Architectural Features**

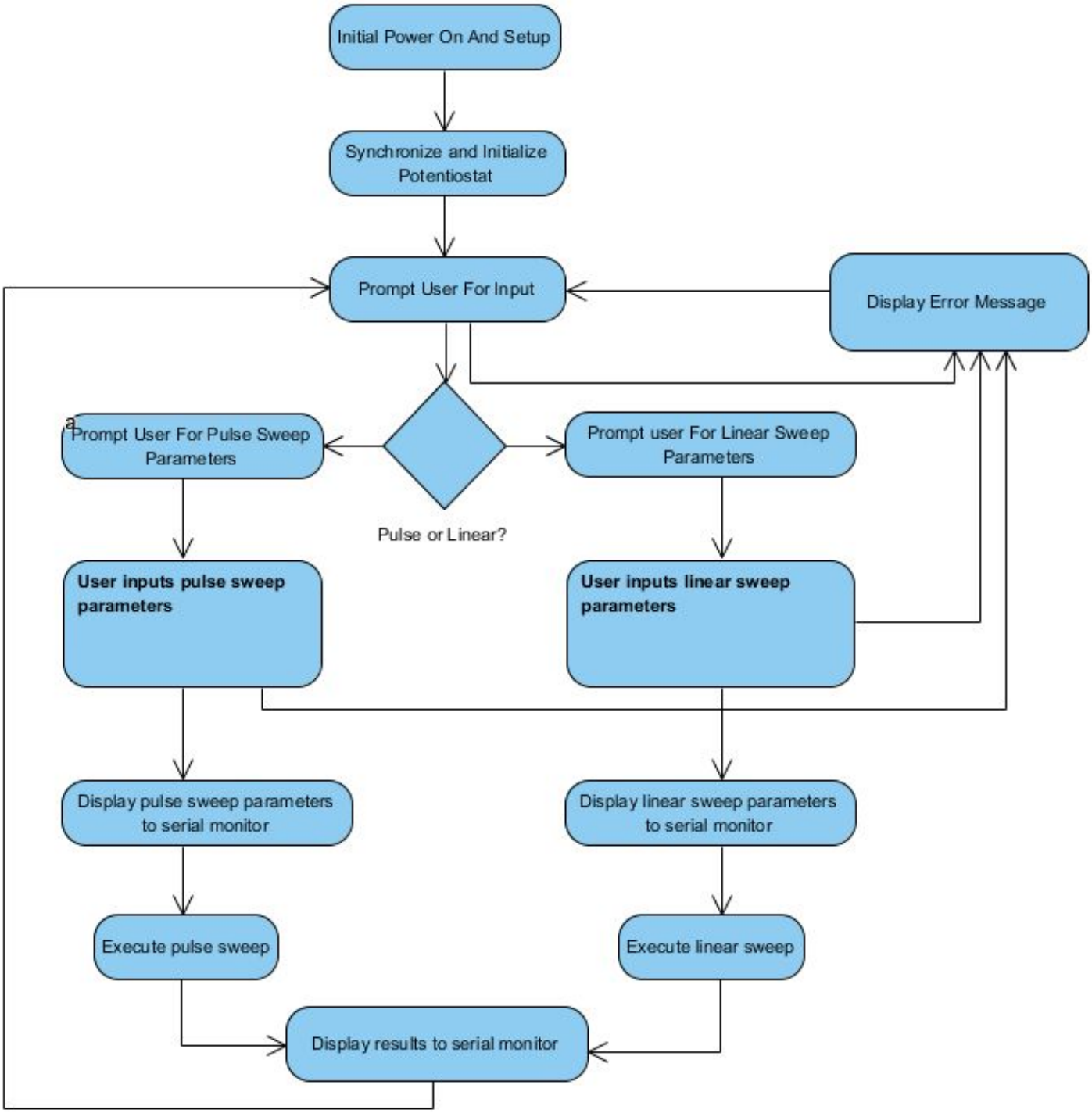


**Figure 23: Software Flow Diagram Of Arduino Mega**

In order to create an extensible software solution that can easily be configured, extended, and revised for future development and application, the software used to program the microcontroller for this project is in a library titled PotentiostatLibrary. The PotentiostatLibrary consists of two files: cpp (source); header file (file declaration). The cpp (source) file contains the function definitions and the header is where all function declarations are defined. As opposed to storing all these functions, member variables, and headers in one Arduino IDE file, the design team believed that condensing these functions into a library, made the software more organized, accessible and easily refactorable.

## Function Definitions

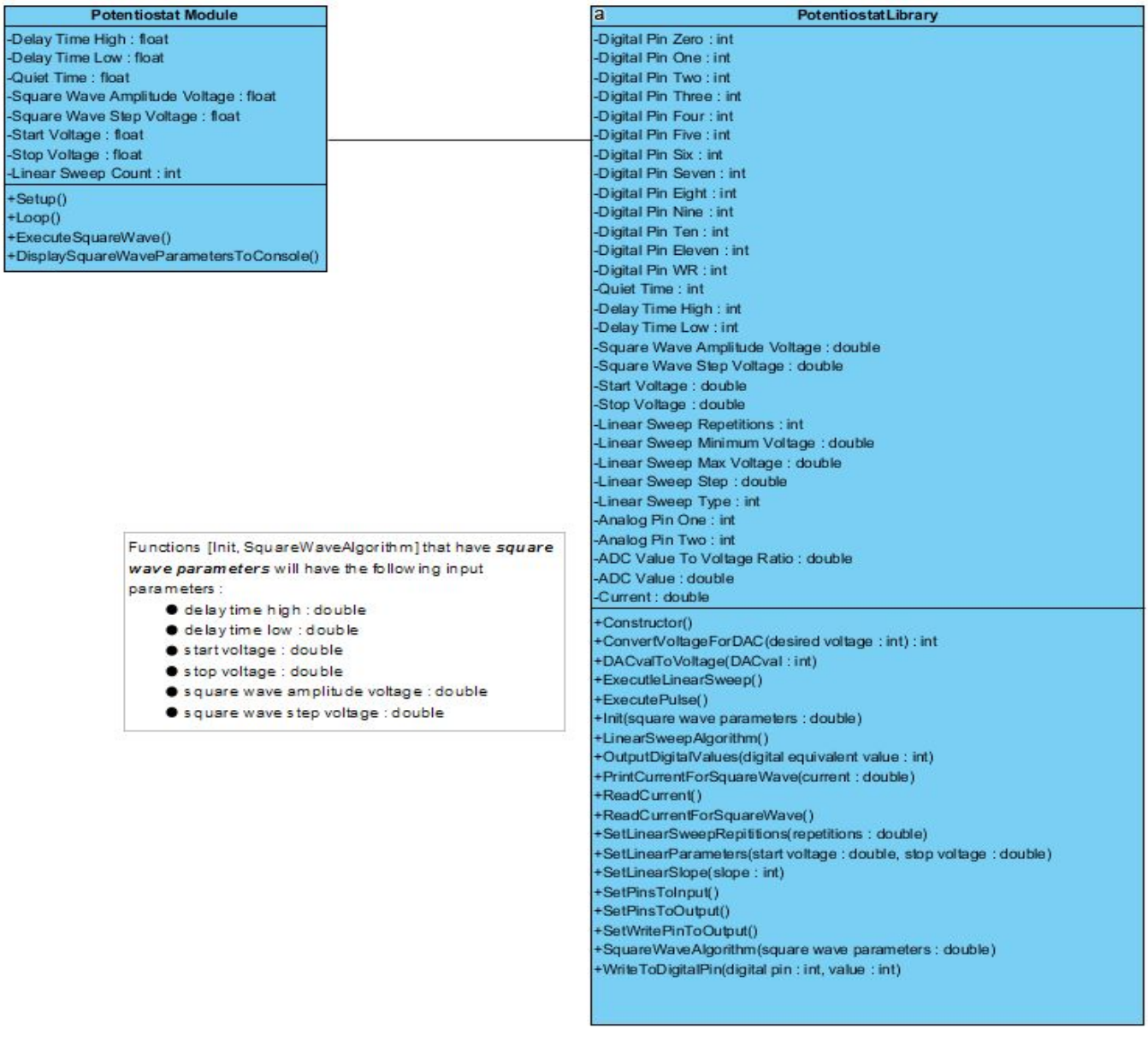


**Figure 24: UML Diagram For Arduino Mega**

The Potentiostat library has total of 30 private member variables, and 19 functions. The executable Arduino file Potentiostat Module contains 8 private member variables and 4 functions.

Constructor: This function initiates and creates a potentiostat object.

ConvertVoltageForDAC: Calibrate the voltage. Referring to analog circuitry, it is a common occurrence where the input values do not always equate to the expected output values. There is a percentage or accuracy error that is tied to the nature of the hardware that is being used in the design. The parameter for this function is the desired voltage, and the return parameter is the calibrated voltage. Both of these voltages are of type integer.

init: This function is used when a pulse wave is selected. This function will set and instantiate all the private member variables required to generate a pulse sweep, and then execute the pulse sweep algorithm by calling the nested function executePulse(). The parameters for this function are all the input parameters required for a square wave: delay time high (ms); delay time low (ms); quiet time (ms); square wave amplitude voltage (v); square wave stop voltage (v); start voltage (v); stop voltage (v).

SetPinToOutput: This function sets GPIO pins [Digital Pin Zero to Digital Pin Eleven] to output. These pins will output either a logic high value of 5v or a logic low value of 0v to the 12 bit external Digital to Analog Converter (DAC).

SetWritePinsToOutput: The private member variable PINPWMR is an isolated signal that is used to signal writing and waiting for the DAC. Before the DAC can read the current input or read the voltages being outputted by the Arduino Mega GPIO pins, there is a GPIO pin on the Arduino Mega titled mDigitalPinWR that toggles reading and waiting. This function sets the mDigitalPinWR to logic high outputting a voltage of 5v to DAC. The DAC will interpret this logic high and will then read the values being outputted by the Arduino Mega. This write pins ensure that all the 12 GPIO output pins have their logic levels assigned and is ready to be received by the external DAC. The write pins provides a very miniscule delay, that is long enough to calibrate and ensure that the data being received by the DAC is correct.

WriteToDigitalPIns: For each entry (bit) in the size 12 array (12 bit) check the value and output either a digital low or digital high signal. If the value for this entry is 0, output a digital low (0v). If the value for this entry is 1, output a digital high (5v).

OutputDigitalPins: Given that an integer is a 32 bit number in C++, to sync with the external DAC the integer has to be converted into a 12 bit binary equivalent. This function converts an integer (voltage) into a 12 bit binary data structure. This data structure is a size

12 array, where each index of the array contains a value of 0 or 1. Each index of the array represents a position in a 12 bit binary value. Each value outputs a voltage to the 12 bit external DAC. To convert an integer into a binary equivalent, extensive bit shifting is used.

Given a 5 volts / 4096 steps = 1.2 milli volts
1.2 millivolt == 000000000001 == 1
2.4 millivolt == 000000000010 == 2
3.6 millivolt == 000000000011 == 3

| Digital Pin Eleven [GPIO 35] | Digital Pin Ten [GPIO 34] | Digital Pin Nine [GPIO 33] | Digital Pin Eight [GPIO 32] | Digital Pin Seven [GPIO 31] | Digital Pin Six [GPIO 30] | Digital Pin Five [GPIO 29] | Digital Pin Four [GPIO 28] | Digital Pin Three [GPIO 27] | Digital Pin Two [GPIO 26] | Digital Pin One [GPIO 25] | Digital Pin Zero [GPIO 24] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2^11 | 2^10 | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

**LinearSweepAlgorithm():** The linear sweep algorithm contains the source code to execute a linear sweep. The function begins by checking whether the linear sweep will begin at a max value and decrease to the minimum value, or begin at the minimum value and increase to the maximum value. To differentiate these two conditionals, the variable mLinearSweepType is used to indicate the nature of the initial slope of the first few points of the sweep. Once a sweep is selected, the user defined initial voltage will be outputted by the GPIO pins and transmitted to the external DAC. This will output that voltage to the solution being tested. The program will then enter a loop and continuously increment or decrement 1 mv to the start voltage until it reaches the user specified stop value. By default the nature of the linear sweep algorithm will travel from the user specified start value to the user specified end value, then return to the user specified start value. This is illustrated by figure 25.

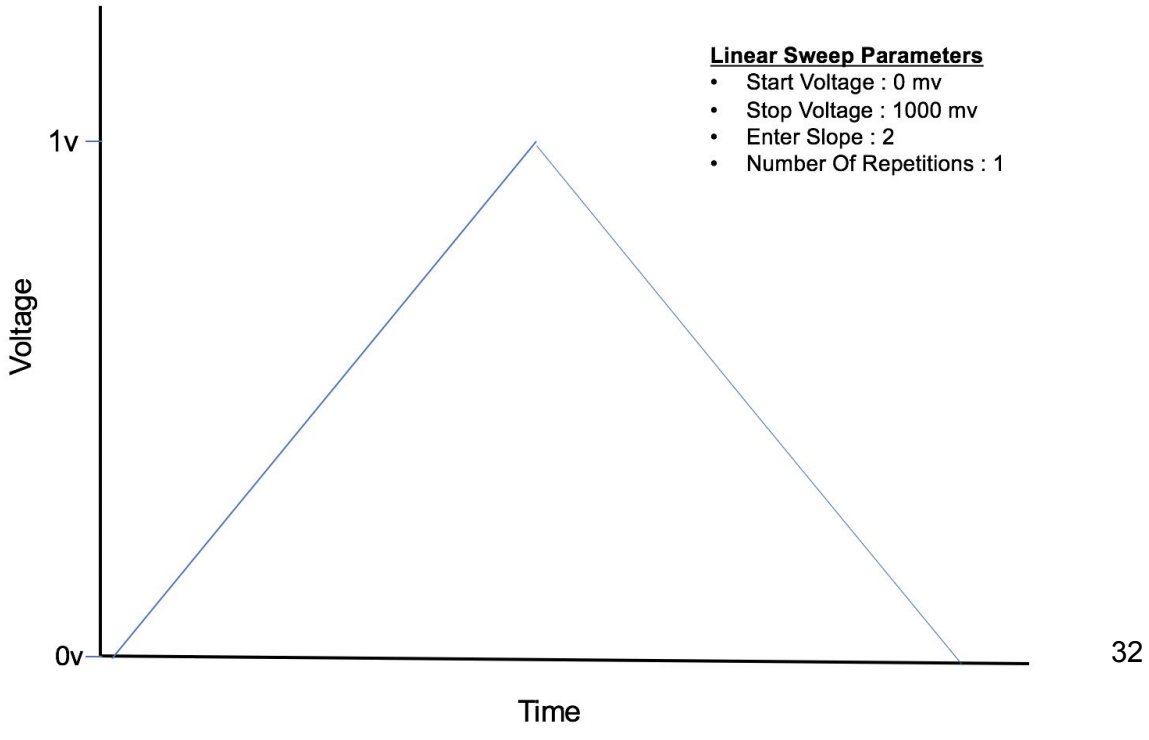

Figure 25: Increasing Linear Sweep

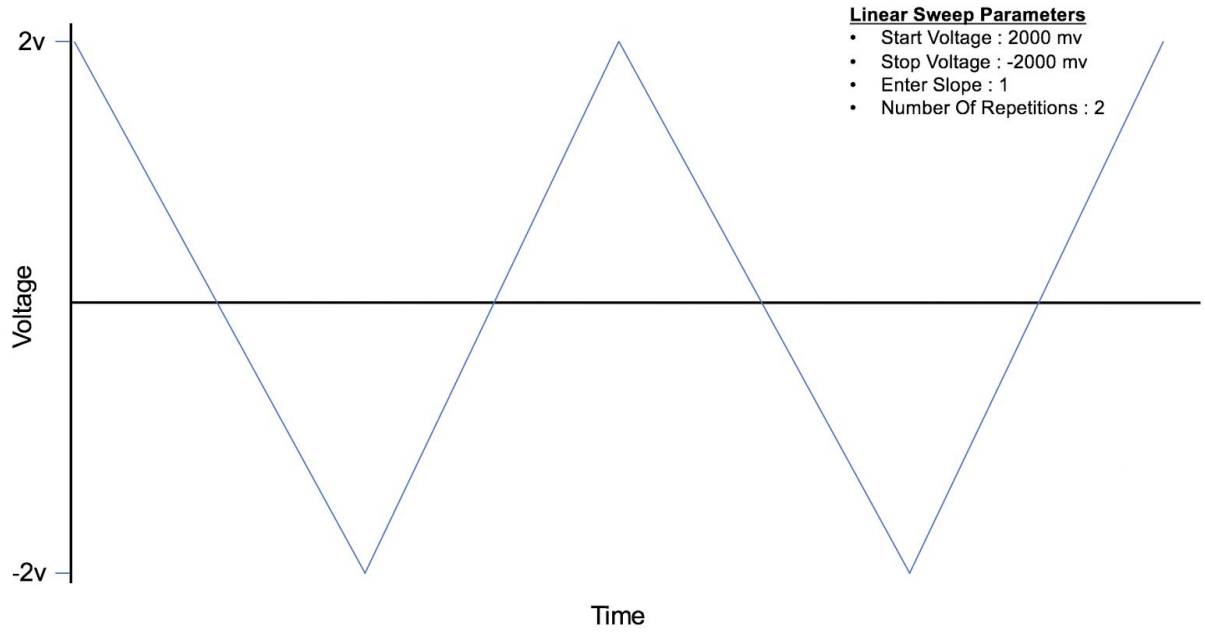


Figure 26 : Decreasing Linear Sweep

The linear sweep algorithm will execute the linear sweep based on the number of repetitions that were inputted by the user.

executeLinearSweep(): Execute the linear sweep and continuously run it for the specified number of repetitions.

executePulse(): This function will execute the pulse sweep. It will first delay for the user input in the quiet time, then it does a function call to the squareWaveAlgorithmn. After the squareWaveAlgorithmn is completed, there will be a 5 millisecond delay, and then the program will restart and re-prompt the user with the initial prompt.

squareWaveAlgorithm(): The parameters for this function are the user input square wave parameters. This function creates the increasing part of the square wave. Each iteration of the for loop generates a pulse with amplitude specified by the user. After each iteration the step is incremented and the next pulse will start at that step. The sample current will be collected during the delay time high then the delay time low of the sweep. The differences between the two currents, current high and current low, will be computed then outputted to the serial monitor where it will be plotted relative to the voltage.

**readCurrent:** This function will read the voltage being applied to the external analog to digital converter. This external analog to digital converter is an Adafruit_ADS115 and it communicates with the Arduino Mega via I2C (inter-integrated circuit) communication. The function will read the current being outputted from the external ADC and then output it to the serial monitor which will then be received by the labVIEW interface.

**readCurrentForSquareWave:** This function reads the raw ADC value from the Arduino pin and converts it into a current using a calibration equation.

**printCurrentForSquareWave:** This function prints the current in a format such that it can be read and accepted by the labVIEW interface program.

**DACvalToVoltage:** This function takes a DAC value as an input, and returns the corresponding voltage that should appear at the output of the circuit.

## Embedded Software User Interface

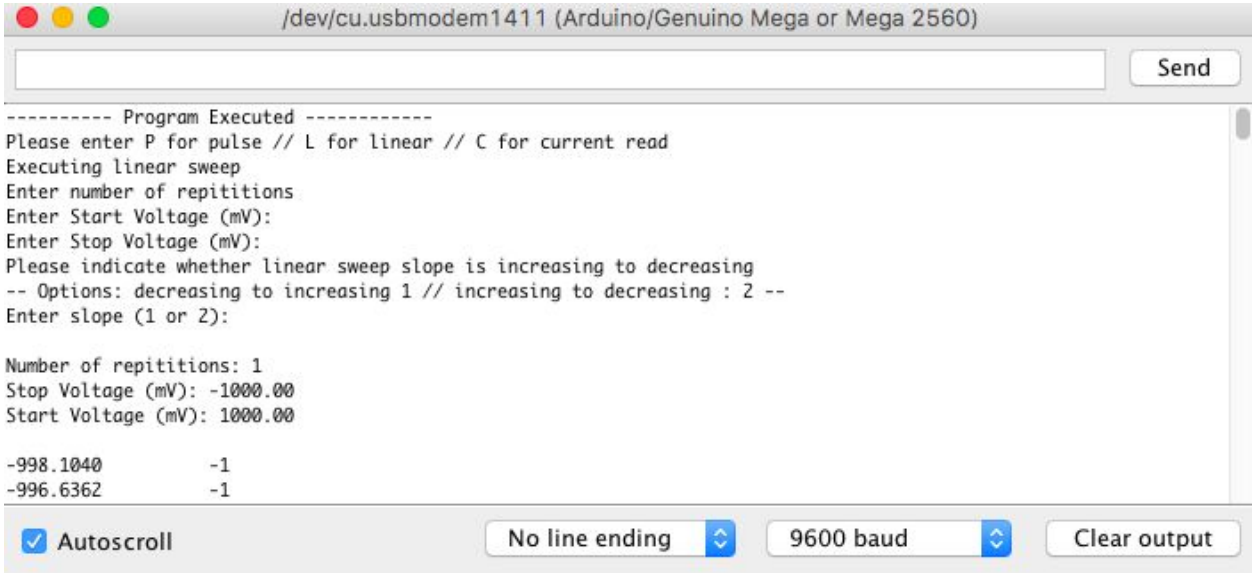


**Figure 27 : Linear Sweep Interface**

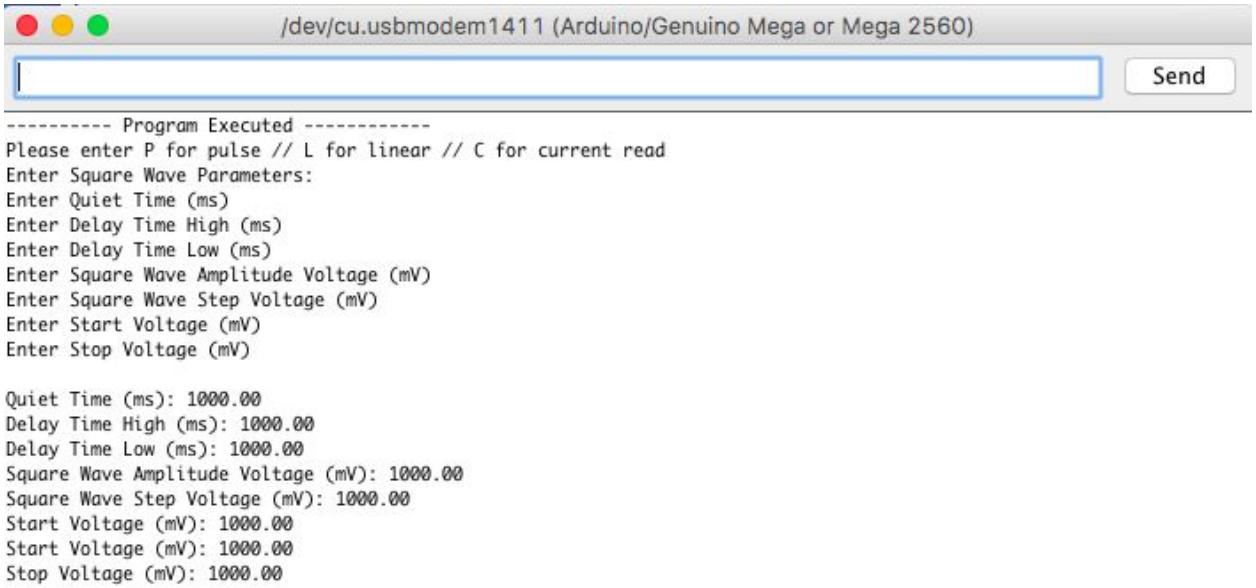


**Figure 28: Pulse Sweep Interface**

The Potentiostat Library supports user keystroke inputs, and can display values to the Arduino serial monitor. It has an integrated user interface that supports full duplex synchronization with the labVIEW user interface. The Arduino user interface is an unappealing solution as it is lacking in design, however it meets our specifications and can be used if LabVIEW is unable to be installed.

## Software Design: Overview

The software application was created to be the user interface of the potentiostat which could communicate with the Arduino and transmit data asynchronously. The software can be break down into multiple parts:

- Software Selection
- Data Communication
- Waveform Display
- File Management
- Software UI

Because the Data communication and Waveform Display are running simultaneously, there are two parallel loops to control the Data Communication and Waveform Display. That means the multi-thread function will be used in the project.

## Software Design: Software Selection

At the beginning of the project, we decided to use MFC in the C++ language by using Microsoft Visual Studio. MFC is a library that wraps portions of the Windows API in C++ classes, including functionality that enables them to use a default application framework. The preliminary software interface built by using MFC shown in the Figure below.

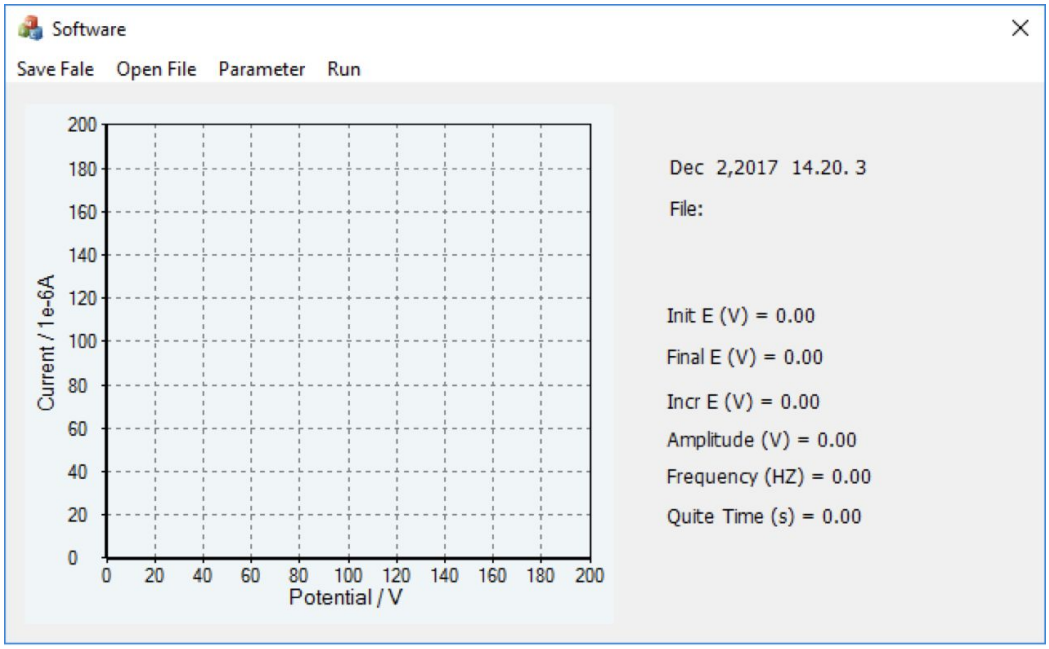


**Figure 29: Software Interface**

However, our software team discovered that building the software solution by using the MFC platform would likely be a risk. That is because many libraries that are used in MFC are outdated and their functionalities are likely to be limited and incompatible to our design. In addition, we may have needed to spend tremendous time writing protocol for communication between software and hardware because none of the team members had experience on that part. These obstacles would have probably resulted in a delay in product delivery, so we wanted to change our platform to keep up with the schedule and ensure effectiveness.

Our secondary choice for software development platform was LabVIEW: Laboratory Virtual Instrument Engineering Workbench. LabVIEW is a system-design platform and development environment for a visual programming language from National Instruments. The reason we chose LabVIEW was because it has many built in libraries and existing data communication protocols between software and hardware that we could directly use. By using LabVIEW, we could not only deliver the solution on time, but also save numerous time that could be used for debugging and perfecting our project.

## Software Design: Data Communication

The data communication was a very important part in the project. The data communication is between the microcontroller and the software. The microcontroller used in the project is Arduino Mega 2560 and the software used in the project is LabVIEW. The main functions of the data communication are configuration, read and write. The communication protocol used in the project is serial communication. Figure 30 is an overview of the read and write architecture.

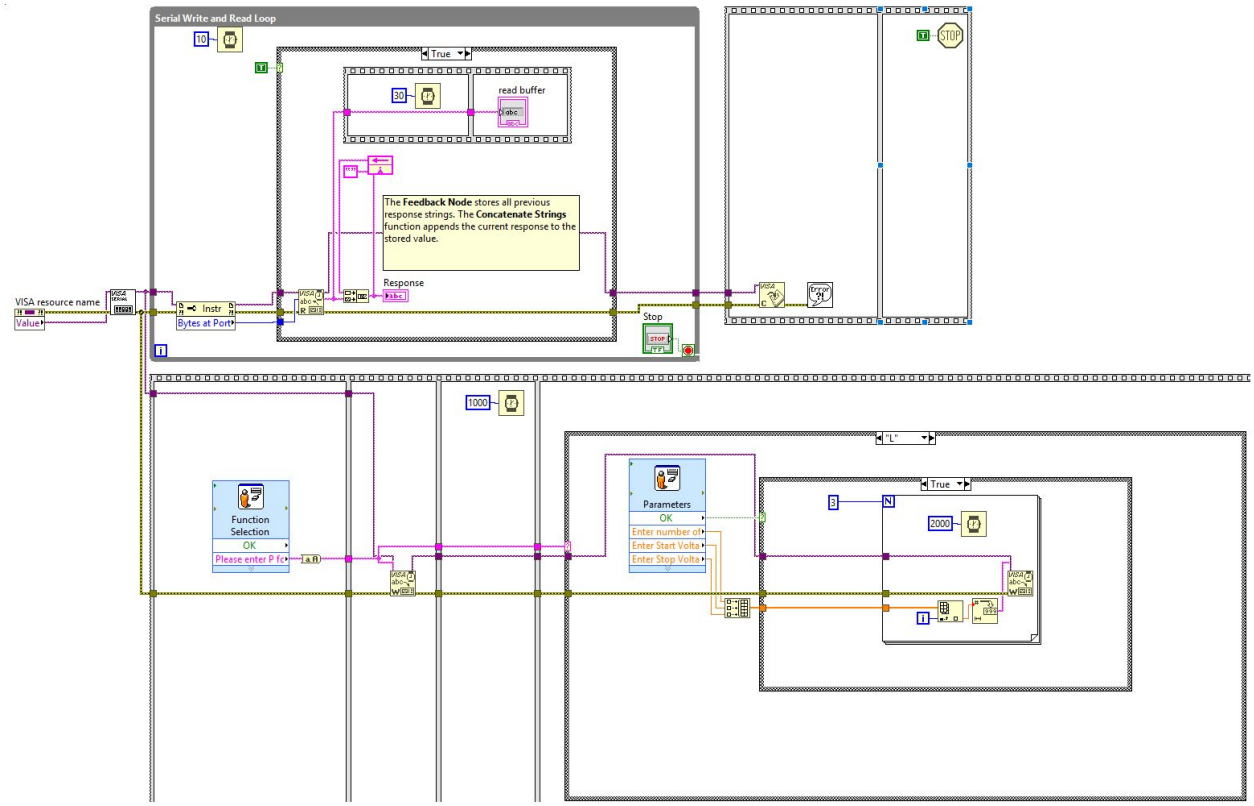


Figure 30: Overview of the read and write structure

The read and write structures are in different blocks; that means the read and write function execute simultaneously. Read function will not bother the write function.

**Configuration:** To configure the serial port, VISA Serial function was necessary. VISA stands for Virtual Instrument Software Architecture which is a standard for configuring, programming, and troubleshooting instrumentation systems comprising GPIB, VXI, PXI, Serial, Ethernet, and/or USB interfaces. VISA Serial's core function is using for establish the connection between Microcontroller and LabVIEW. Figure 31 shows the inner structure of VISA serial given by National Instruments. In addition, figure 32 shows how to configure serial communication in this project.

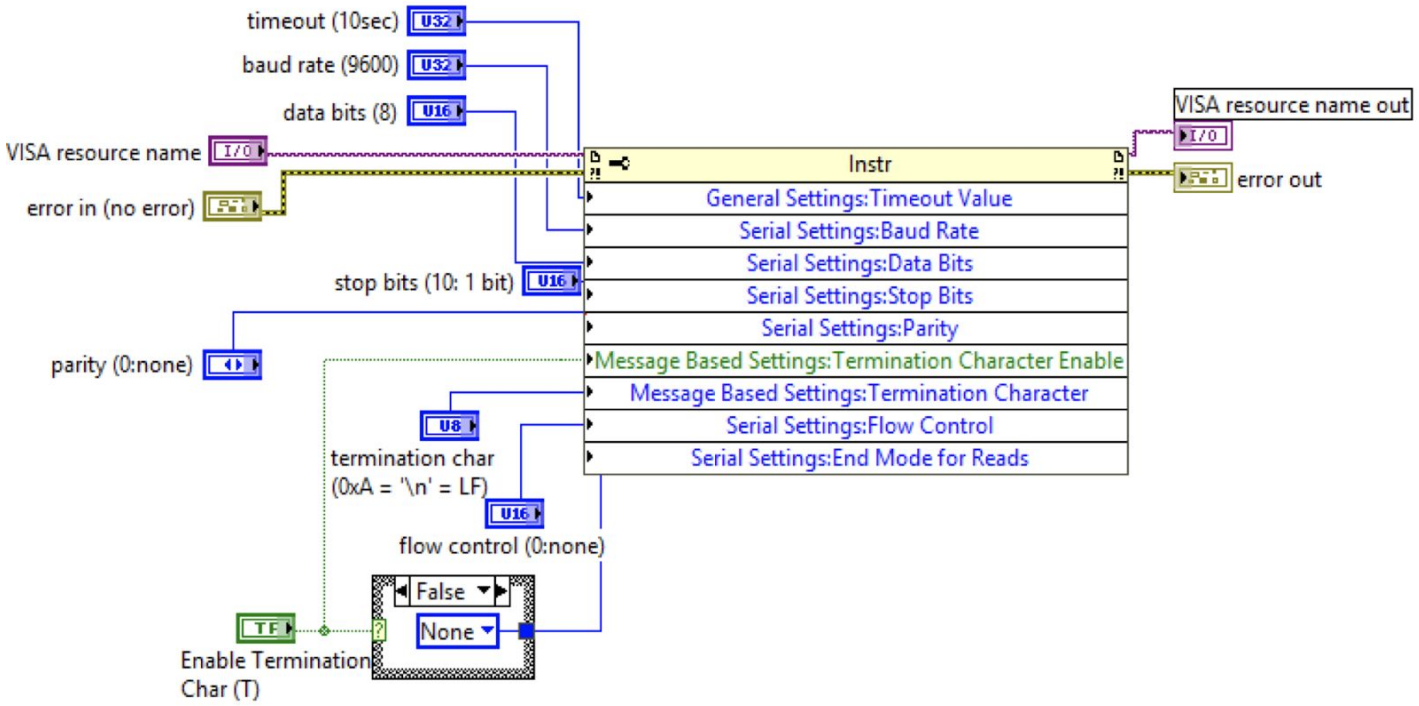


Figure 31: The inner structure of VISA Serial

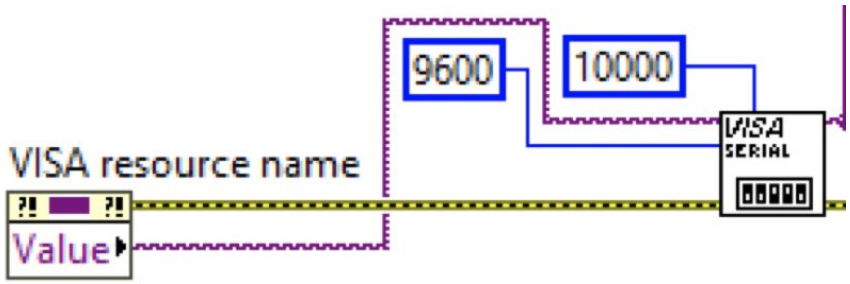


Figure 32: Serial port configuration

VISA Serial has various functions. Functions used in the project are VISA resource name, baud rate, timeout and error. Other functions remain as default. When the serial device is plugged into the computer, the computer will generate a serial port number for the device, and the users can select the device by using the VISA resource name function. The VISA serial block can recognize the device by receiving the data from VISA resource name. Then, each microcontroller has its own baud rate. Therefore, selecting correct baud rate in the VISA Serial function will make the serial port send the data correctly. The project uses 9600 baud rate for Arduino Mega. Timeout is a way to avoid the deadlock of the program. Once the program doesn't respond in 10000ms (10s), the VISA Serial will automatically close the serial port and send an error message through error out.

**Read:** After initializing the serial port, the next step is reading the data from the microcontroller. The core function use in read is VISA Read. VISA Read will read the specified number of bytes from the device or interface specified by VISA resource name and returns the data in read buffer. Figure 33 is the loop for reading the data from Microcontroller.

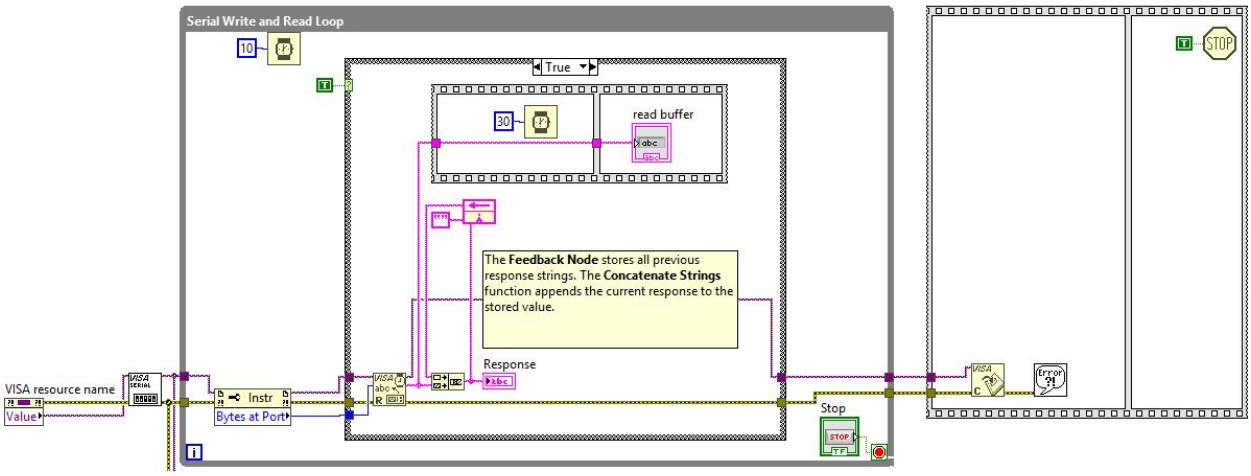


Figure 33: The loop for continuous reading

The VISA Serial function will initiate the serial port and send the data to the next block. The block VISA Read will receive the data and store all messages into a read buffer. The block between VISA Serial and VISA Read is VISA Bytes at Serial Port. It will return the number of bytes in the input buffer of the specified serial port. This block will limit the data transmission between VISA Serial and VISA Read. By using this block, VISA Read will only read and store the data whose bytes are larger than 0. Because serial communication will send the data to the serial port line by line, every time the read buffer receives new data the old data will be cleared. To store the old data to the memory, using bundle to create an array is the best choice. Each time new data is read into the buffer, the old data will store into the array. It will keep the old data and continuously read the new data. The outer loop is a while loop. The while loop is controlled by a Boolean switch. Once the switch is on, the while loop will stop. The while loop is used for continuous read. After the while loop is stopped, the program will run the VISA Close. VISA Close will close all used serial port and release the occupied port. So by using the loop with read function, all data can be read and stored into the array continuously. There is a wait timer in the loop because the serial communication needs some time to process the data, and using a wait timer helps avoid reading too fast.

**Write:** Another objective in the project is write the data back to the Arduino. The core function needed to do this is VISA write. This function can write the data from the write buffer to the device or interface specified by VISA resource name. Whether the data is transferred synchronously or asynchronously is platform-dependent. When you transfer data to or from a hardware driver synchronously, the calling thread is locked for the

duration of the data transfer. Depending on the speed of the transfer, this can hinder other processes that require the calling thread. However, if an application requires that the data transfer as quickly as possible, performing the operation synchronously dedicates the calling thread exclusively to this operation. Figure 34 is the write block.

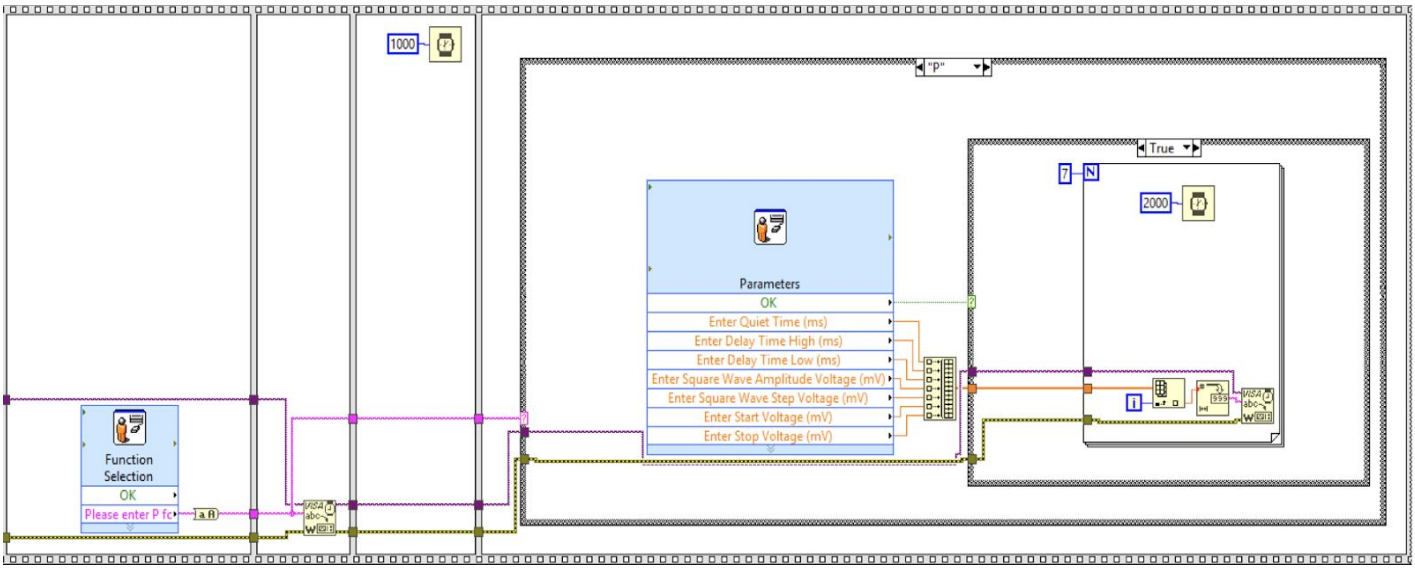


**Figure 34: The write block for writing the data into the MCU**

The write block is created by using the flat sequence structure. Each part is separated by the flat sequence frame. The program will execute from left to right. The first frame prompts the user to input the parameters(Function selection). Then it goes to the next frame. The second frame will write the parameter into the Microcontroller. Because the serial port is a low speed communication port and the microcontroller needs to process for a while, using the wait function in the next frame is necessary to ensure the data transmits completely. After the wait, a Boolean function is used for error handling. If the parameter is not acceptable, the program will stop. If the parameter is correct, then it goes to the next step. In the case structure, a new prompt window will show up. After the user enters the parameters for Linear Sweep or Square Wave, those parameters will be bundled together and output as an array, and the array is controlled by the Index Array function. Because of the same reason, the speed, after each parameter is written into the Microcontroller, there should be a wait. Moreover, because the VISA write only accepts the String input, using the Number to Decimal String function is needed to convert it.

## Software Design: Waveform Display

The Waveform Display will display the measured current and voltage onto the XY graph. Users can analyze the data from the waveform. Figure 35 is the overview of the waveform display structure.

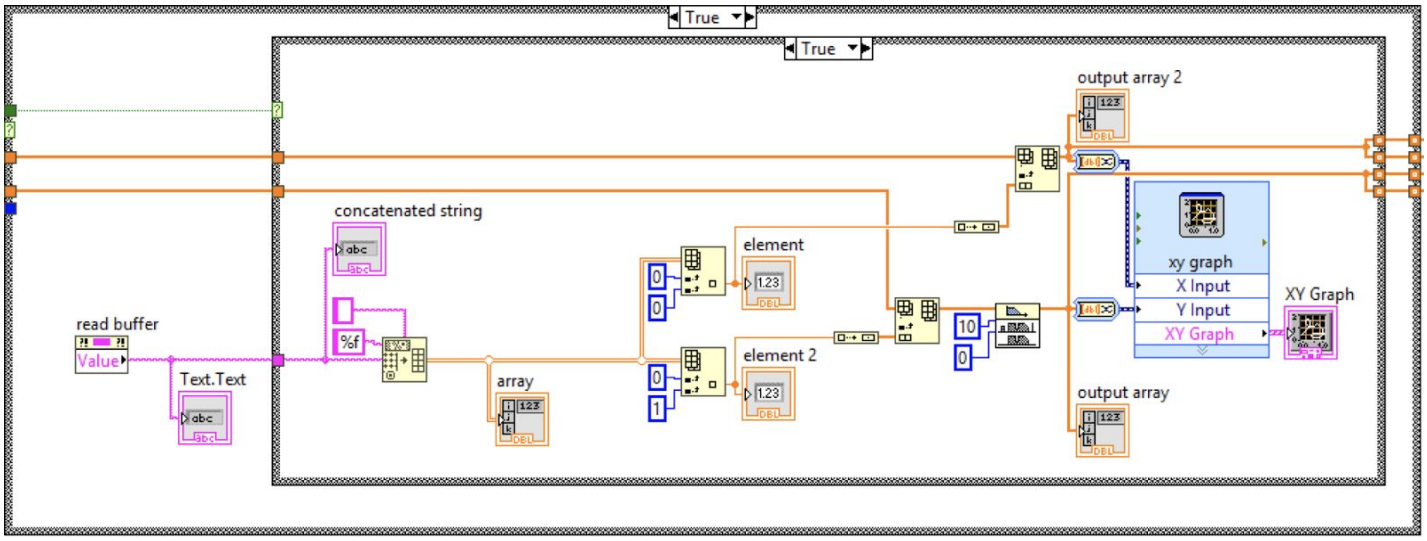


Figure 35: The overview of the waveform display

All data displayed on the graph comes from the processed Raw Data.

**Raw data split:** The data read from the microcontroller are the raw data. Raw data cannot be plotted on the graph directly, so the first step is to split the raw data for the future data processing. The figure shows the split structure.

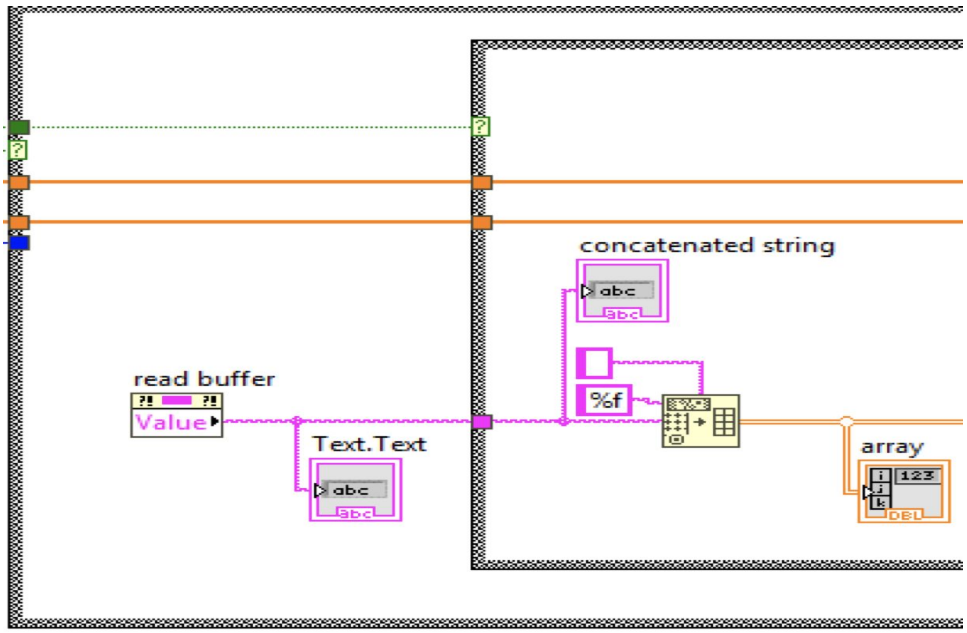


Figure 36: The split structure

The read buffer stores the data returned from the Microcontroller. This read buffer is a value property node of the previous read structure's read buffer. Figure 37 illustrates part of the previous read structure.

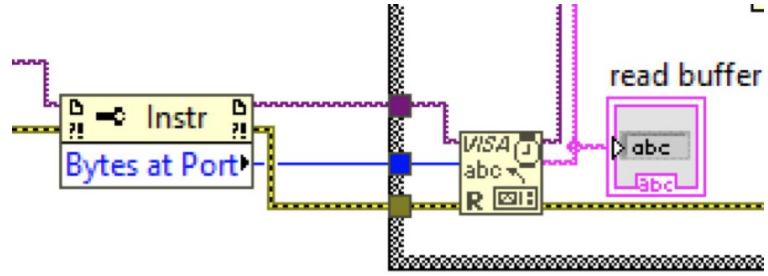


Figure 37: The part of read structure

The data in the read buffer is the raw data, the format is voltage with space and then followed by the current. To split the raw data into two numbers, using spreadsheet(String to Array) is the best choice. Spreadsheet can convert the spreadsheet string to an array of the dimension and representation of array type. This function works for arrays of strings and arrays of numbers. In the project, space is used as delimiter and %f is used as format string.

Data processing: After splitting the raw data, the current and voltage will be stored as the array in the memory. To fetch the data from the array, Index Array is used. The figure shows how to fetch the data from the array.

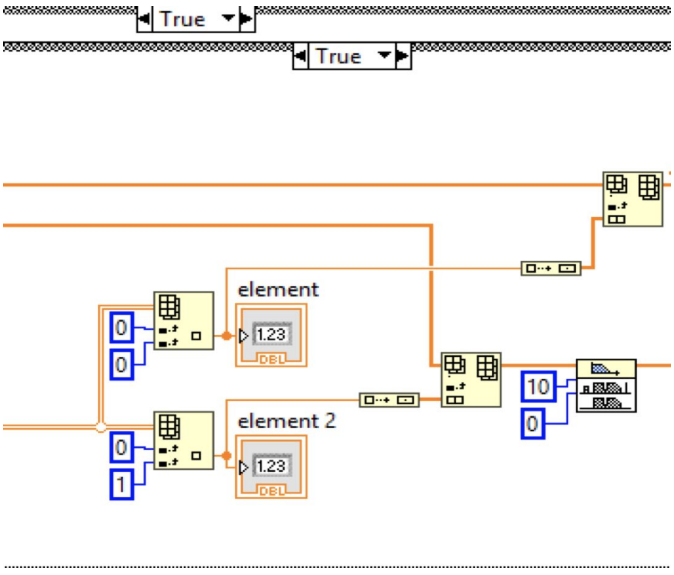


Figure 38: Data processing

There are two Index Arrays used in the structure: one element stores the current and other stores the voltage. Voltage is generated from Microcontroller directly, but the current is read from the external analog signal; that mean there exists noise in the data. To lower the noise in the data, a filter is used to do the noise canceling after fetching the data. The processed data will be sent to the next step.

**Data Display:** After data is processed, raw data have been processed into two elements: Voltage and Current. Current will be displayed as the Y-Axis and Voltage will be displayed as the X-Axis. The figure below shows the XY-graph structure.

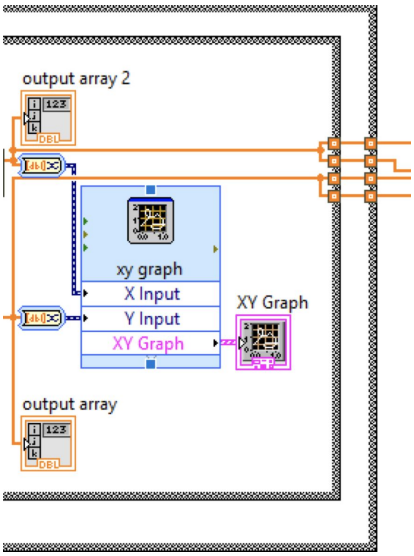


**Figure 39: The XY-Graph structure**

Current and Voltage will be sent to the XY-Graph. To avoid data error, there are two dynamic data converters between X input and Y input. X input is the voltage data and Y input is the current input.

## Software Design: File Management

To store the data for future using, the designed software has File Saving and Opening function.

**File Saving:** In the file saving part, the software is designed to save the waveform data into two separate text files. One is for x-axis value and the other is for y-axis value so that we have two text files generated simultaneously.

**File name setting:** First, we need to set the path for file saving. There is a string that is user-input, the file path, using for directing the file location. The saved files are named by user-input prefix respectively with string label 'x' and 'y' at the end of the name. Prefix is a user-input string for identifying the saving file. It is part of the saved file name.There is a concatenate Strings block to concatenate the user-input prefix and 'x' and 'y' label to generate the file name.

**Building the save path:** There is a user-input path that user can select existing or create new folder to save the files. Once the path is selected and prefix is filled, clicking the run button will execute the build path block for generating the file.

**Writing data:** After the Building path block, we implement a Writing data Block. It has an external link connected to the result data from the waveform and should be able to write the result data into the new created file.

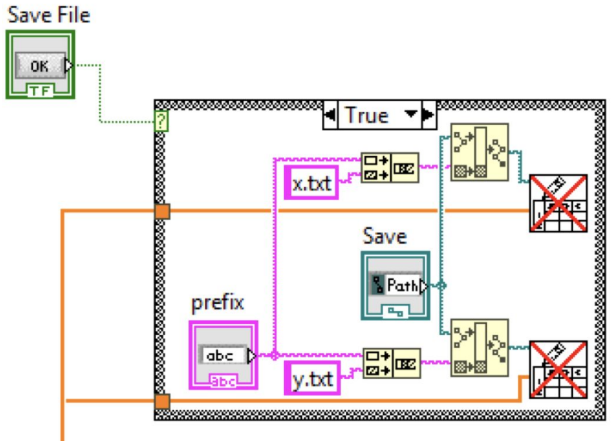


Figure 40: File Saving Block Diagram

The figure above is the overview of the file saving block

**File opening (Reproducing Waveform):** Our software should be able to open two saved files according to user-input prefix and user-selected file path and then plot the waveform based on the data in the saved file.

**File opening and Data reading:** File locating is very similar as the **File name setting (Please see File Saving: File name setting)** The only difference is the File opening case structure has a Reading Data Block instead of a Writing Data Block. The Reading Data Block will read the data after the file is correctly located.

**Reproduce Waveform:** After the data is read, the data will be send into two empty arrays for x and y value respectively. The Waveform block has two input links for arrays and plot the waveform based on the x array value and y array value.

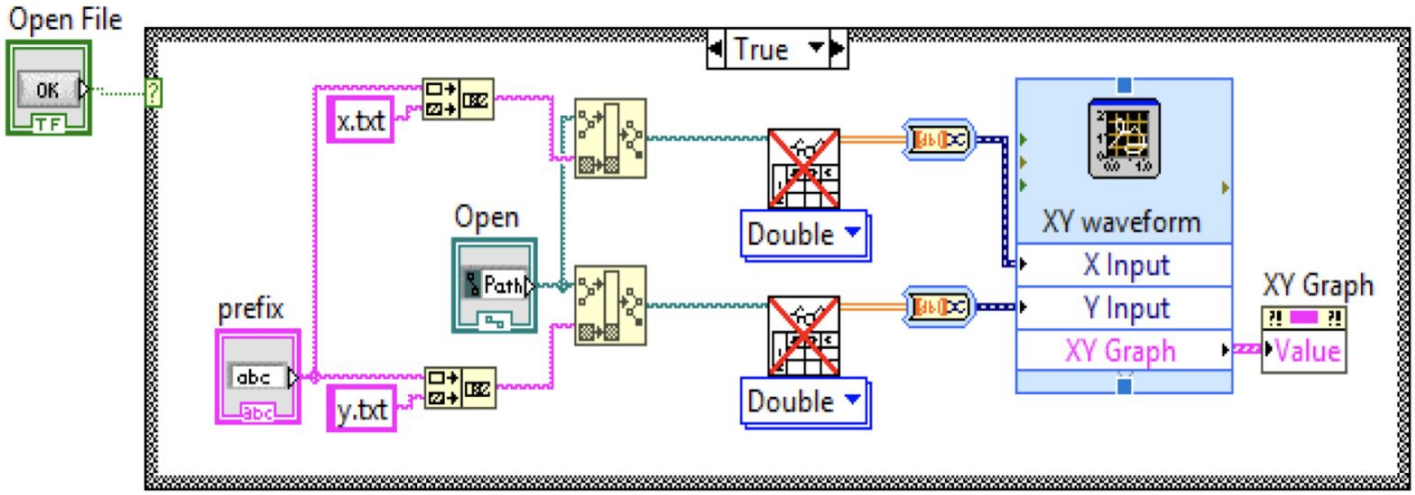


Figure 41: Open File Case Diagram

## Software Design: Software UI

Software UI is designed for users. Users can input the parameters to start the program.

**User-input parameter:** For starting the measure, we will require user to enter the type of measurement and parameters. These data will be send to the hardware (microcontroller) via array.

**Function Selection:** It is constructed by Prompt User Block. User needs to enter either 'P' or L" for selecting measurement function. Inputting other strings will trigger an ' Invalid Input' error message to the user which is implemented by Display Msg Block. After error triggered, the software will stop.

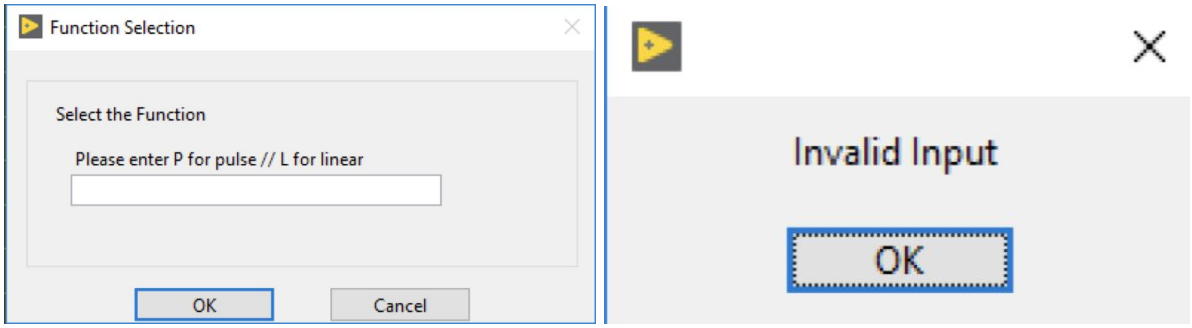


Figure 42: User Prompt UI and Error Msg UI

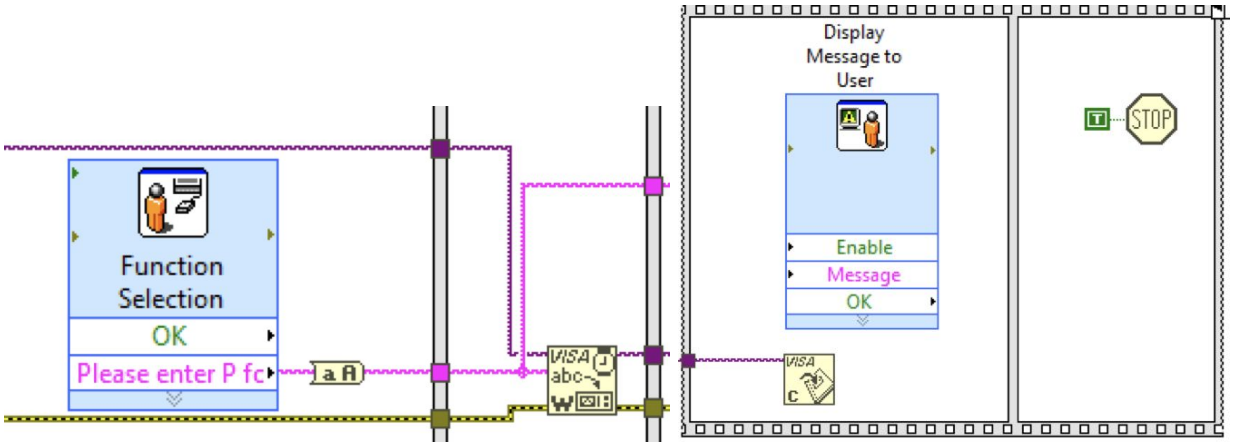


Figure 43: Function Selection User Prompt and Display Msg block diagram

The figures above are the blocks of Error handling. The execute order is from left to right.

**Pulse Wave Prompt:** If the user enters string 'P' in the Function Selection window, the Prompt User Block shown in **Figure 44** will ask the user to enter the square wave parameters. In this UI window, users are allowed to input numbers only for preventing errors. Clicking "Ok" on the parameters UI windows will package the data into an array and send the input parameters to the hardware (microcontroller). Clicking "Cancel" will stop the software.

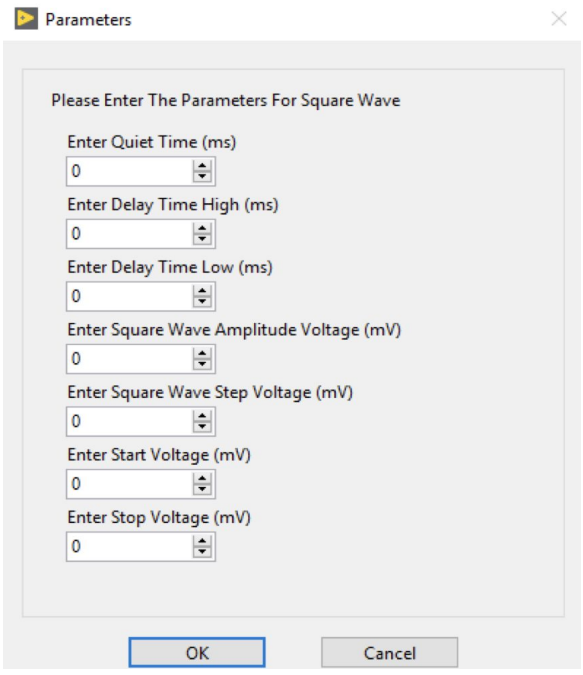


Figure 44: User-Input Parameters for pulse wave UI

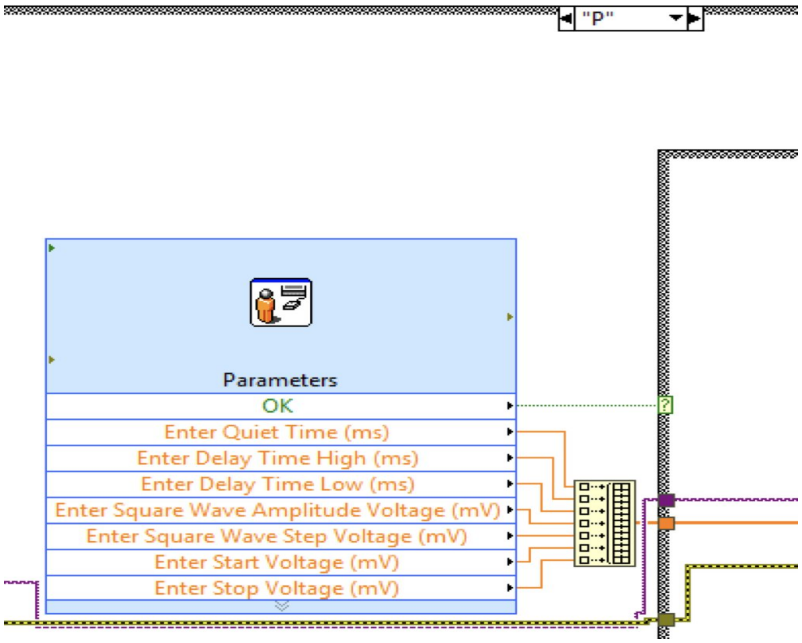


Figure 45: User Input Parameters for Pulse wave Block Diagram

All parameters are listed in the figure.

Linear Sweep Prompt: The construction of Linear Sweep Prompt is similar to Pulse Wave Prompt. Both of them are constructed by the User Prompt Block. The Linear Sweep prompt UI will ask the user to input the number of repetitions, start voltage, and stop voltage. In this UI window, users are only allowed to input floats / numerical values. Clicking Ok on the parameters UI windows will package the input parameters into an array and send input parameters to the hardware (microcontroller). Clicking Cancel will stop the software.

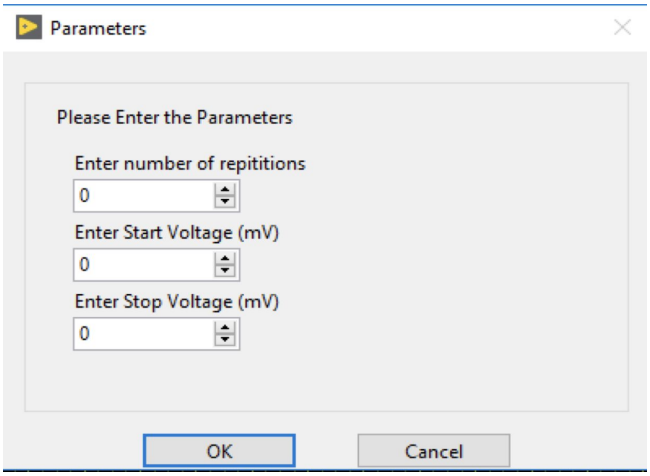


Figure 46: User Input Parameters for wave User Interface

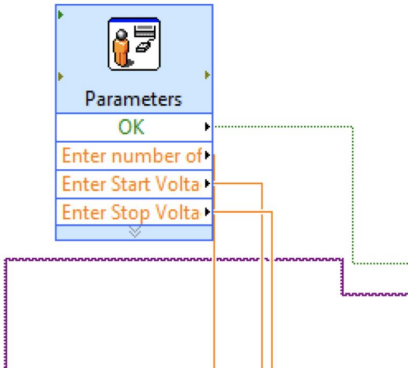


Figure 47: User Input Square Wave Block Diagram

## Software Design Complete Solution View

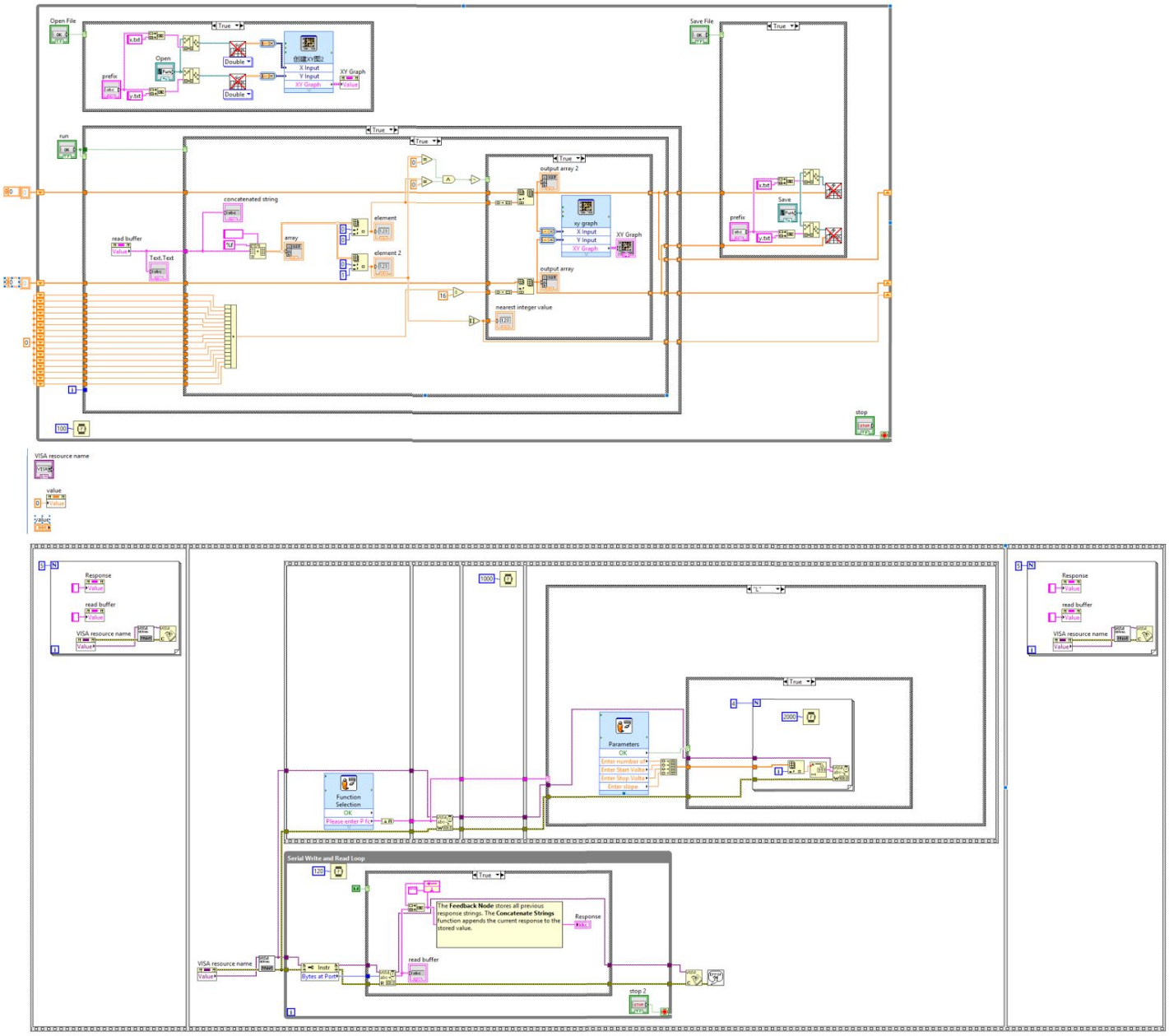


Figure 48: LabVIEW Interface Block Diagram Full View

## Software Design UI

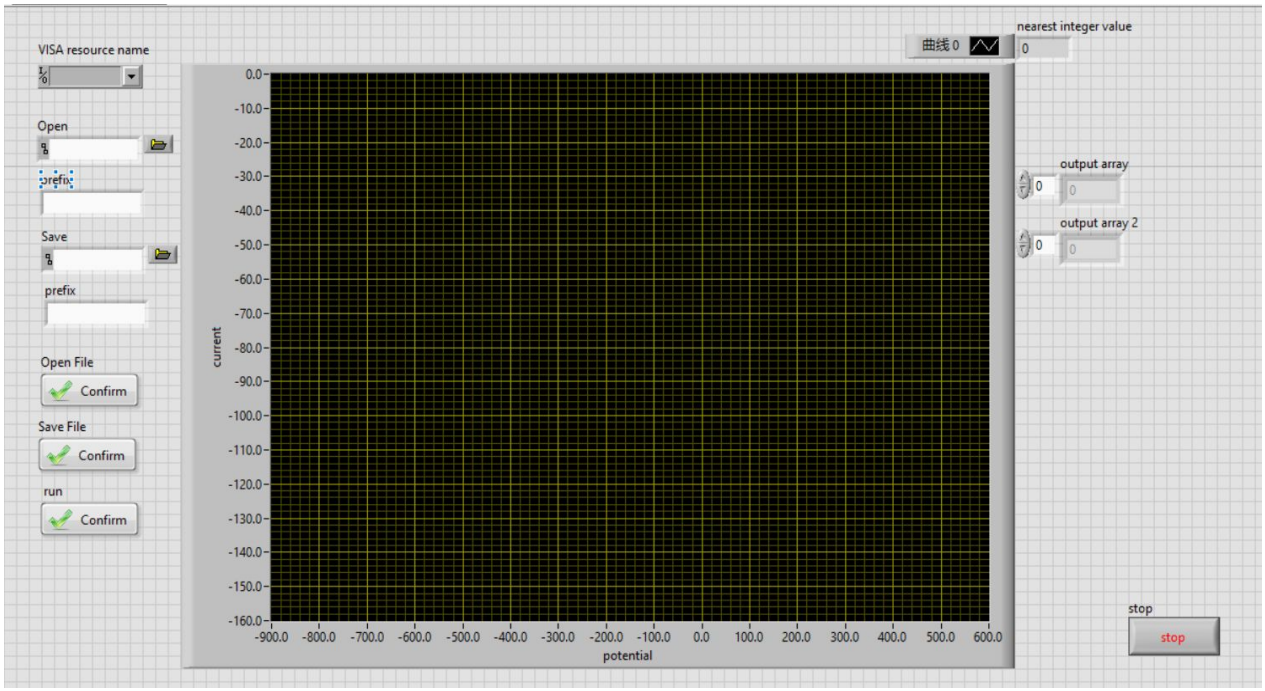


Figure 49: LabView User Interface

# Chapter 4: Test Data with Proof of Functional Design

## Unit Testing of Hardware

After building the voltage driver and calibrating the software, we were able to verify the accuracy of the voltage at the working electrode when specified by the user. In testing this, a feedback resistor was used between the working and reference electrodes to simulate the behavior of a solution under testing. Voltages at the working electrode were measured using the multimeters found in the ECE 480 lab.

| Voltage Input by User (V) | Measured Voltage at Working Electrode (V) |
|---:|---:|
| -2.000 | -1.9994 |
| -1.000 | -0.9991 |
| 0.000 | 0.0021 |
| 1.000 | 1.0015 |
| 2.000 | 2.0006 |

**Figure 50: Table of Voltage Output**

After building the current sensing hardware and calibrating the software, we were able to verify the range and accuracy of the measured current generated at the counter electrode. Currents were read using the multimeter found in the ECE 480 lab. Note that the potentiostat is most accurate between about -80uA to 20uA. This is likely due to the nonlinear characteristics of the on-board Arduino ADC that start to appear as the ADC approaches its maximum input.

| Current Read by Potentiostat (uA) | Current Measured in Lab (uA) | Current Read by Potentiostat (uA) | Current Measured in Lab (uA) |
|---:|---:|---:|---:|
| -79.42 | -80.0 | 67.06 | 80.0 |
| -49.87 | -50.0 | 47.01 | 50.0 |
| -20.19 | -20.0 | 19.94 | 20.0 |
| -10.16 | -10.0 | 9.66 | 10.0 |
| -5.22 | -5.0 | 4.87 | 5.0 |
| -0.77 | -1.0 | 0.78 | 1.0 |

**Figure 51: Table Of Current Read**

## Square Wave and Linear Waveforms

We were able to successfully generate linear and square waveform voltage sweeps at the working electrode. The linear sweep start and stop values can be set by the user to be anywhere from -2V to 2V. The square wave can also be adjusted by the user to set parameters including quiet time, start voltage, stop voltage, amplitude voltage, step voltage, delay time high, and delay time low. The following images were taken with an oscilloscope probe connected to the working electrode.

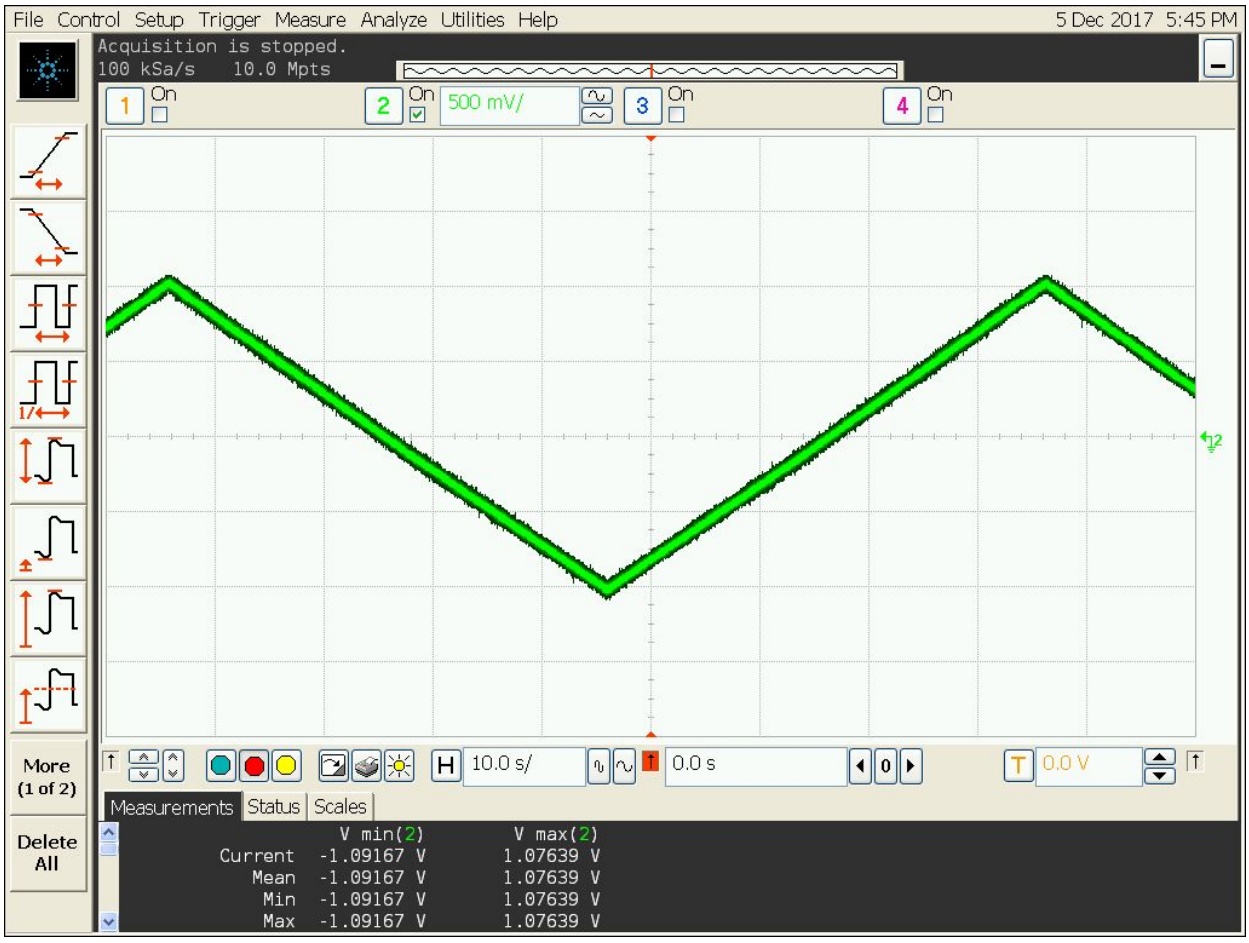


Figure 52: Linear Sweep from -1V to 1V

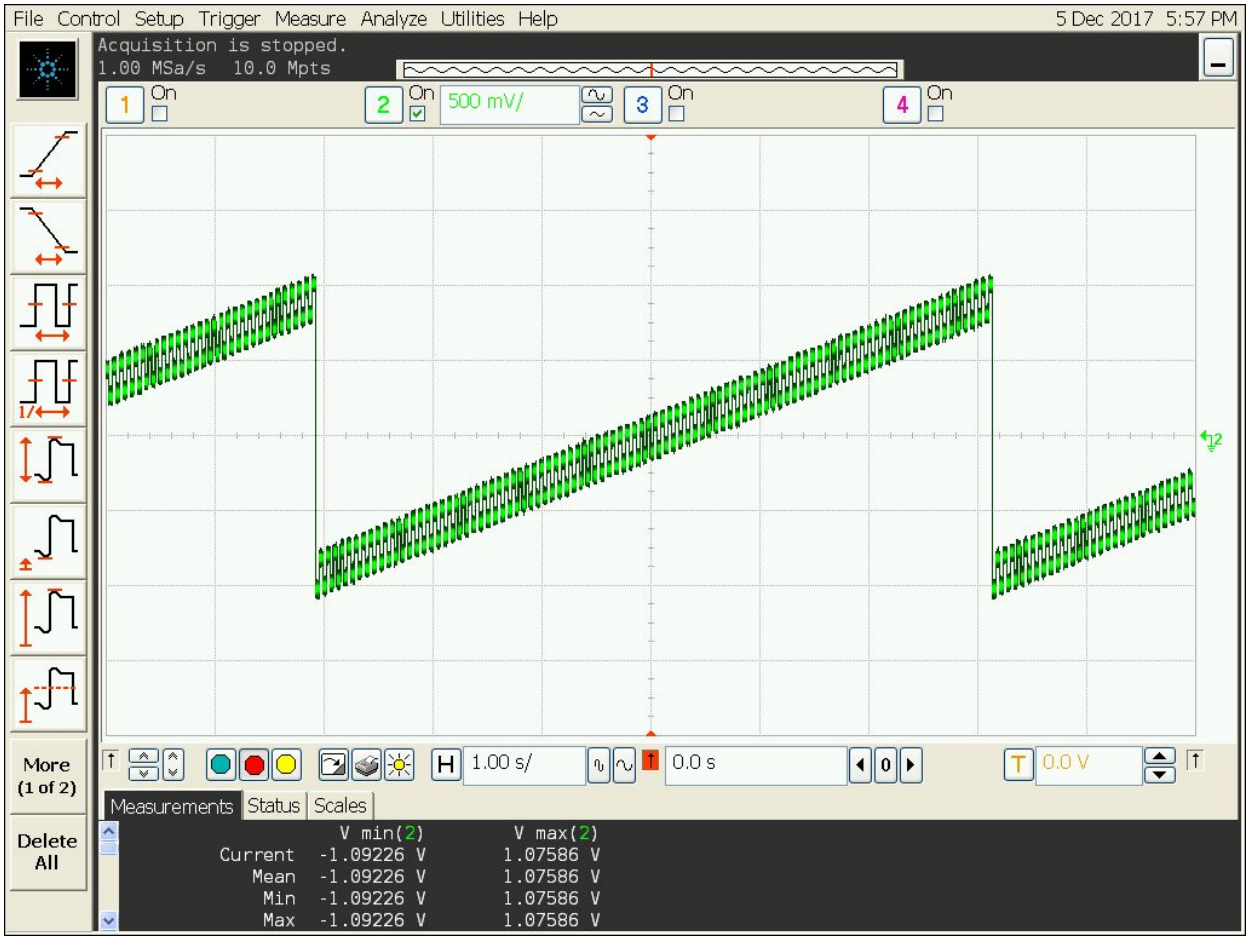


Figure 53: Square Wave Sweep from -1V to 1V with Delays set to 30ms, Amplitude = 200mV, and Step = 20mV

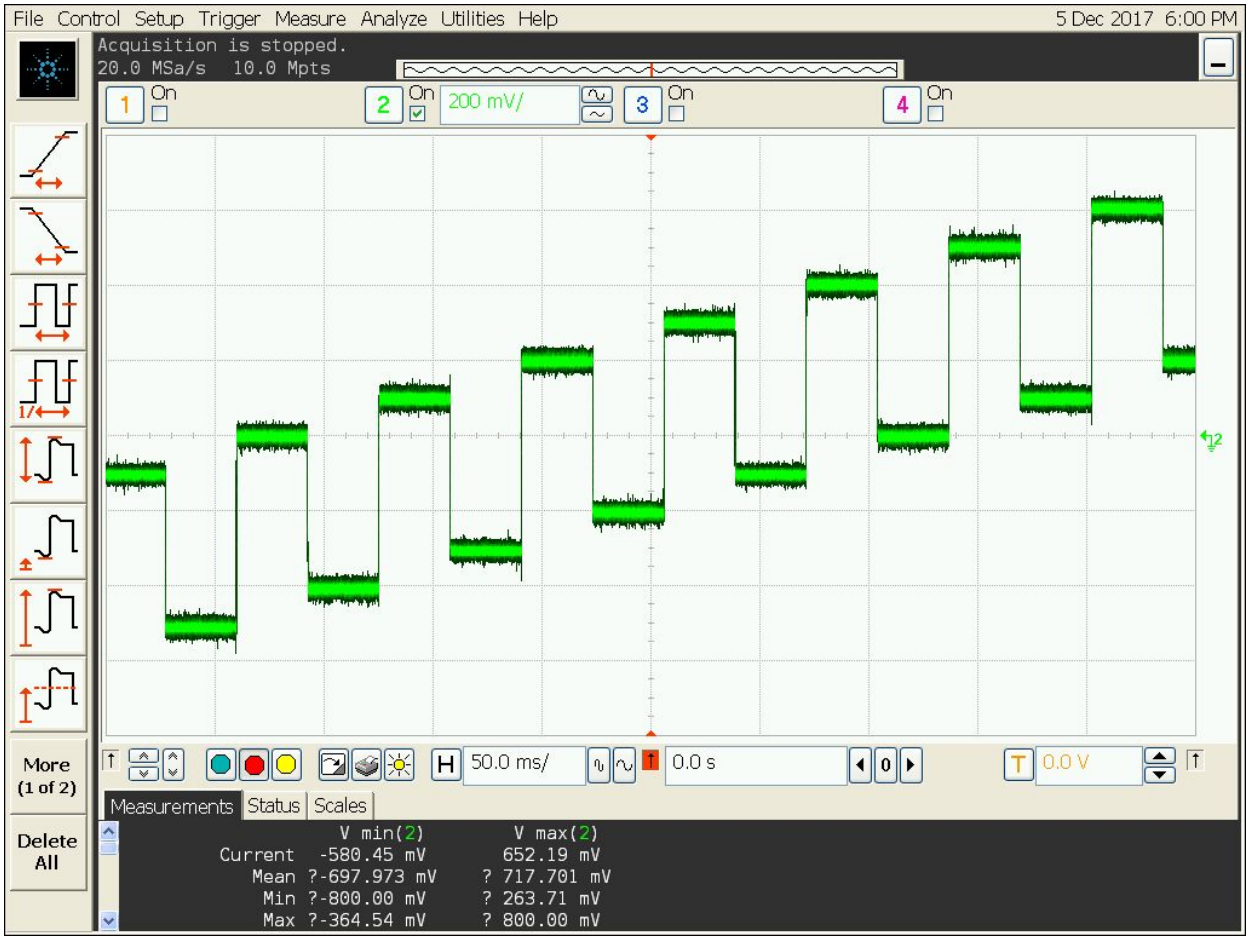


**Figure 54: Square Wave Sweep Zoomed In**

## Prototype

An initial prototype was built and used to verify that our hardware, embedded software, and software interface were all able to synchronize with each other. In addition, this prototype was used to show whether our design would meet the project specifications. Although this prototype utilized a protoboard as opposed to a fully complete printed circuit board, this allowed quick revisions and modifications to the analog circuitry. We tested our design using a potassium ferricyanide solution. Our tested prototype was able to execute linear sweep and square wave sweep as defined in the requirements, although returned data from the square wave sweep was suspected to be erroneous. Voltammograms constructed using our prototype are included in the Simulation and Testing section of this chapter.

## Printed Circuit Board

The printed circuit board was constructed by the Michigan State University College Of Engineering Tech shop. Unfortunately, our printed circuit board was not able to perform as we initially speculated. This is most likely due to some shorted traces, traces that go underneath the op-amps, and the lack of a consistent ground plane. The PCB shown below is the third iteration that the team created, after fixing some issues in previous iterations including traces that were too close together and vias that were too small to be drilled through.

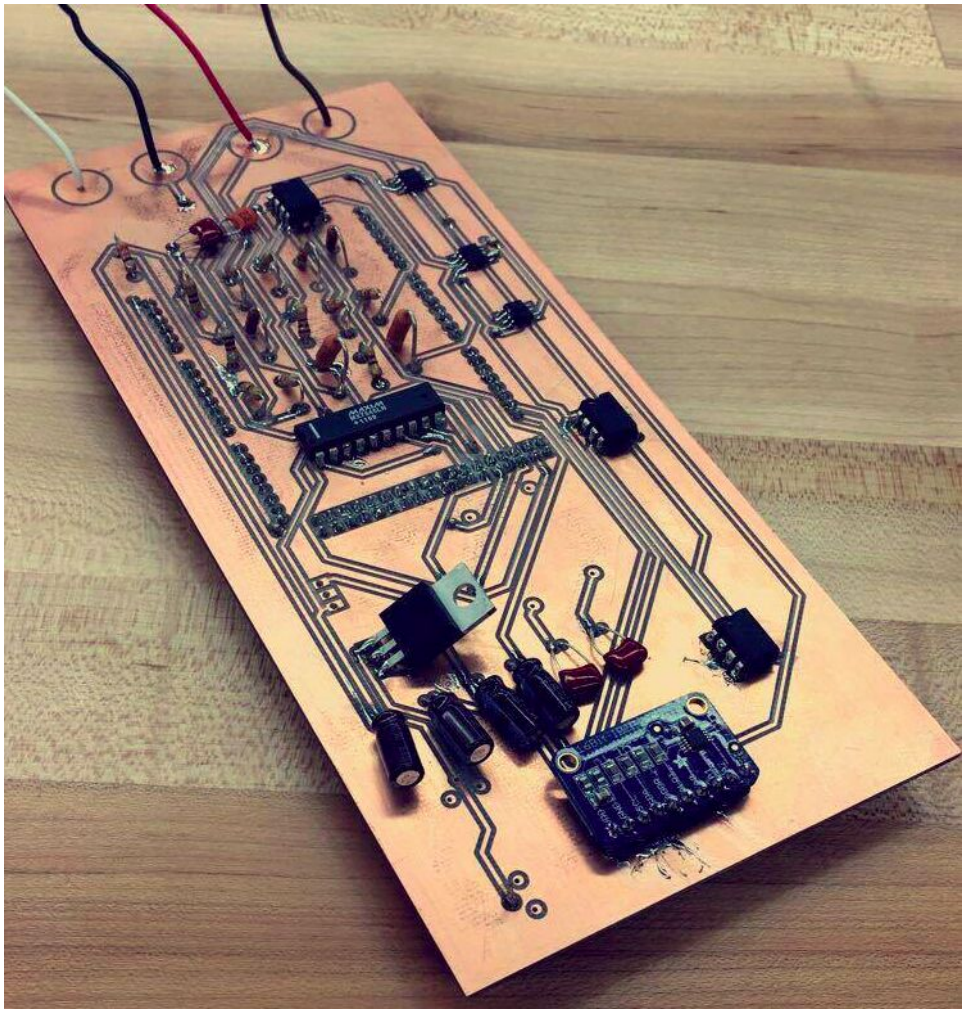

**Figure 55 : Printed Circuit Board**

## Simulation And Testing

Below is a linear sweep voltammogram captured by the team's potentiostat. The captured voltammogram features some erroneous data, as well as gaps in the returned data that are likely due to data lost during transmission. This likely occurred due to too much data being sent to LabView than LabView can process.

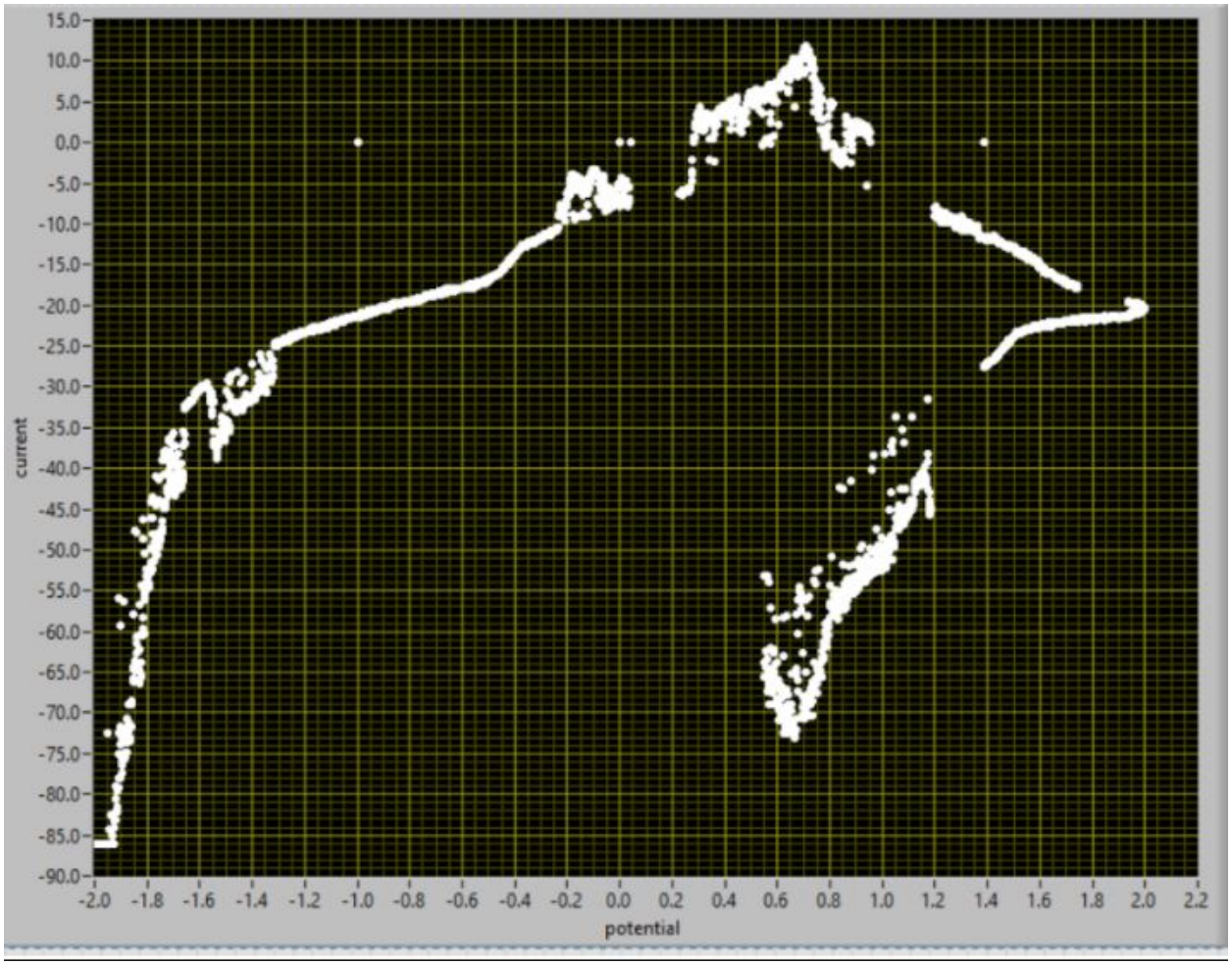


**Figure 56: Linear Sweep Voltammogram with Some Lost Data**

Square wave voltammograms were also created by using the square wave sweep feature. However, although the voltammogram looks similar to what we would have expected, the ADC is clipping the value of the actual current coming from the counter electrode. The square wave feature was not able to be fixed in time for Design Day.

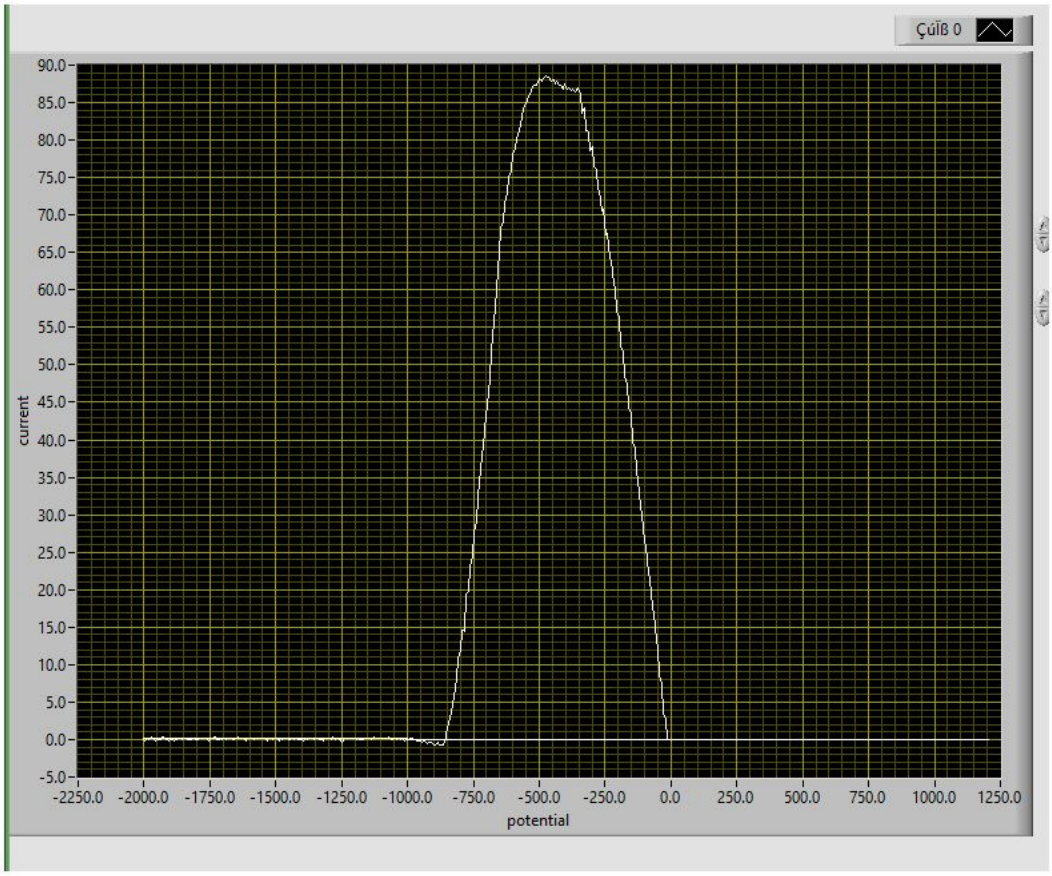


Figure 57: Square Wave Voltammogram

## Design Specification Table vs. Performance

| Specification | Protoboard Prototype |
|---|---|
| Execute Linear Sweep | Yes |
| Execute Square Wave Sweep | Can generate waveform, cannot read back current correctly |
| Minimum Output Voltage Step Size : 10 mv | Yes (min step size of ~1mV) |
| Output Voltage Range [-2v , 2v] | Yes |
| Average time to complete one cycle : 25 seconds | Yes, if parameters set correctly |
| Minimum Current Sensitivity : 1 microamp | Yes, with some inaccuracy |
| Data must be outputted to an external platform (laptop, phone) | Yes |
| Returned data can be graphed | Yes, but with some communication errors |
| Data can be returned into a human readable format and stored in a text file | Yes |
| End device is handheld and portable | Yes |
| Device can be powered by an external power supply | Yes |
| Data and simulation is 90% accurate to CH Instruments Potentiostat | No |

**Figure 58 : Table Of Design Specifications**

In summary, our final design module (hardware, software) met most of the base requirements and specifications that were defined by the facilitator and sponsor at the beginning of our project. However, there are many improvements and additions that need to be implemented prior to anyone using this device as a credible data collection resource.

Unfortunately, our end design did not perform as well as we initially speculated. Ideally this portable potentiostat could be used as a substitute to current potentiostats used in industry. However, our simulation and test results have led us to conclude that there needs to be changes applied to the hardware and embedded software in order for this to occur. Although some of our components and implementation may be sufficient, in order for this device to be practically useful and functional, changes need to be applied in a few areas.

In conclusion, our design is a successful failure. We were able to meet most of the base specifications as defined by our sponsor, but were lacking in a few others. Necessary improvements and design fixes are laid out in the next chapter. Our final design may serve as a solid foundation for any individual or team aspiring to develop a portable potentiostat.

# Chapter 5: Design Issues

## Accuracy

The accuracy of the potentiostat could have been greatly improved if we had been able to get a higher bit ADC to work for current sensing, instead of using the Arduino's onboard ADC. In addition, the output of the voltage driver circuit could be measured using an ADC in order to get a more accurate measure of what the voltage at the working electrode actually is, instead of just trusting that the potentiostat is correctly calibrated. Although we purchased a higher bit ADC to be added to our design, we were not able to get it functioning properly before our final deadline.

## Data Transmission

When transmitting data from the Arduino to the LabView application, data will occasionally be lost, leading to large "gaps" in the voltammograms. The cause of this transmission loss is unclear, but it may be due to data being transmitted to LabView faster than LabView can store and process it. This problem could be solved by sampling less data, or running slower sweeps.

## Latency

There is currently about a 10 second delay between when the voltage sweep is executed, and when the data appears on the LabView plot. This was done intentionally to allow LabView time to process some of the data. Although this issue does not affect the accuracy of the device, it can be frustrating for a user to have to wait 10 seconds for a plot to begin to display. Lower latency between the microcontroller and the LabView application would lead to a much improved user experience.

## Square Wave Generation

The team was not able to get the square wave voltammetry feature able to work properly. When using a test solution to test the square wave feature, it appeared that the current being generated was exceeding 85uA for most of the sweep. This was causing much of the returned data to be meaningless, as the potentiostat was not able to measure the true value of the current, and was instead returning a clipped value. It is not clear if this problem could be solved by increasing the range of the current sensing circuit, or if the problem is due to faulty software.

## Clipped Data Detection

If any data read from the current sensing circuit is clipped, the user has no way of knowing that the data is inaccurate. Having a piece of software that can detect when the ADC is at its max or min value could be used to notify the user that the data being returned is inaccurate.

## Microcontroller

During the research and development process of this project, there were many difficulties determining and integrating a microcontroller that was suitable as an ideal interface platform. Referring to the three microcontroller choices: Texas Instruments; Cypress Semiconductor; Arduino, each microcontroller had significant pros and cons in comparison to each other.

Ideally it would have been nice to integrate the Texas Instruments MSP430G2553 into our final design. Despite it lacking an external DAC, it was an ideal selection since it satisfies all other specifications and requires low power. Low power was a significant pro due to to this device being battery powered.

The Cypress Semiconductor was the most technically advanced and featured internal peripherals which would result in accurate readings. In addition the IDE incorporates LabVIEW compatibility such that integrating this device with the Software created by the ECE 480 Team 11 should be a smooth transition. As speculated in the proposal, the Cypress Semiconductor microcontroller was difficult to integrate and was not integrated into the final design. Due to the limited amount of resources and scheduling constraints, the design team decided to revert to the Arduino Mega.

## Power Supply

For the power supply, the chosen design fulfills the requirements, but due to the unexpectedly high power requirements of the circuit, the batteries tend to die within a few hours. This could be remedied with a different battery system, such as a pack of LiPo cells with associated charging and balancing circuitry. This would allow for a rechargeable system with a much higher battery capacity.

## PCB Design

Although the team was able to design and fabricate a PCB, the team was unable to get this design to work. The PCB design had some shorted traces, traces that went underneath op-amps, and was lacking a consistent ground plane. If these issues were fixed, the PCB would have likely worked.

# Chapter 6: Final Cost, Schedule, Summary, and Conclusion

## Final Cost

The total cost for our project can be found in the detailed budget below.

**Project Cost**

| | Subgroup Descriptions | Budgeted Cost | Actual Cost |
|---|---|---|---|
| PROJECT DEVELOPMENT | Prototype parts | $ 150.00 | $ 134.01 |
| | Misc | $ 50.00 | |
| | **Subtotal** | **$ 200.00** | **$ 134.01** |
| FINAL DESIGN | PCB | $ 80.00 | $ - |
| | Enclosure | $ 10.00 | $ 10.00 |
| | Power Supply | $ 20.00 | $ - |
| | Additional Circuitry | $ 50.00 | $ - |
| | Misc | $ 50.00 | $ 44.08 |
| | **Subtotal** | **$ 210.00** | **$ 54.08** |
| OTHER COST | Other cost | $ 50.00 | $ - |
| | Other cost | $ - | $ - |
| | Other cost | $ - | $ - |
| | **Subtotal** | **$ 50.00** | **$ -** |
| | **Subtotals** | **$ 460.00** | **$ 188.09** |
| | Total Remaining | **$ 40.00** | **$ 311.91** |
| | **Total Spent** | **$ 460.00** | **$ 188.09** |

Figure 59: Final Project Cost Outlined

Due to careful planning and minimal expenses, we were able to stay significantly under budget for the project. Many of the parts required were far cheaper than expected. Also, we were able to use the ECE shop for our PCB removing a significant potential cost from the budget. We could have reduced the amount spent even more if we had known which microcontroller would be the best in advance instead of purchasing some that ended up being unusable or overcomplicated.

## Final Schedule

A minimalistic schedule overview can be found in the figure below.

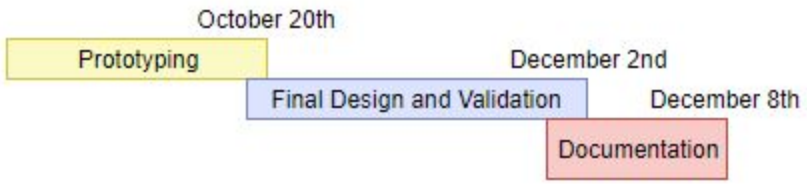


**Figure 60: Final Schedule**

Unfortunately, we were not able to follow the original schedule as closely as we would have liked to. Due to holdups caused by needing to change microcontrollers twice in addition to waiting for parts and getting the Arduino to talk to LabVIEW effectively, we had to push completing the final design to December 2nd. Thankfully, we were able to still complete almost everything we wanted to and finish our documentation on time. This can also be seen in the full size Gantt chart in appendix III.

## Summary

By using the voice of the customer, the design team was able to outline requirements for the project. Following this, we consistently met with the sponsor and facilitator to confirm various aspects of the project and to update progress. The design included numerous unique aspects including hardware design for the voltage driving circuit, power supply circuitry, a printed circuit board, current sense circuitry, an enclosure, extensive embedded code, and an interactive LabView user interface. Many of these aspects presented significant challenges, and the final design is a result of many failed iterations and subsequent design improvements. Although all of these aspects have been completed, there are numerous design improvements that have been considered by the design team. These can be found in the future improvements section of the report below.

## Conclusion

In conclusion, the design team was able to build a working potentiostat while being significantly under budget. The team was forced to make some significant design changes during the semester after many first pass designs did not work as intended, but was able to fix most of the major issues. Although there are still issues with generating voltammograms

when using the square wave sweep feature, the linear sweep feature can be used to generate voltammograms. There are definitely significant flaws with the design, including a possible loss of potentiostat data being transmitted to the LabView application. However, the majority of the project does work as it was intended to, including the user interface, square wave and linear sweep generation, and current reading. The design work completed could serve as a valuable foundation to any other team or person who is interested in building a portable potentiostat.

## Future Improvements

It would be nice if this device could be supported on mobile platforms such as IOS and Android. For this to be successful, a complete mobile application would have to be developed, where full duplex bluetooth communication from the portable potentiostat and the mobile phone would have to be supported. Several system design considerations would have to be taken into account when transitioning from a laptop application to a bluetooth connected cell-phone app. A necessary change would be to add a connection between the positive 9V battery and the Arduino's power input in order to power the board without usb. Also, in order to minimize noise in general, emc shielding should be added to all of the sensitive measurement electronics, especially the precision op amps in order to add a bluetooth transceiver while also minimizing electromagnetic noise.

Another hardware consideration would include redesigning a new PCB with primarily surface mount components and the Arduino's main chipset, the Atmega2560. By doing this, the board size could be greatly reduced resulting in a much smaller overall design. As mentioned in more detail in the power supply section, by adding LiPo cells with a charge controller and balance circuit, battery life could be massively extended as well as adding rechargeability.

Referring to the embedded system, there are additional features that could be programmed and implemented into the design. Considering the Arduino mega is only utilizing only a small percentage (16%) of its total computing power, there are additional internal peripherals and processing that could be added to enhance the performance for the user.

Error handling is a shortcoming that was not fully addressed in the software. Although the software design team did account for common errors such as invalid user inputs, they did not account for other experimental errors such as: removing the electrodes from the solution, removing a connection during simulation, or disabling the power supply during a sweep. Error handling can be added by adding additional conditional statements and conditions to the potentiostat library.

Considering potentiostats are expensive devices and are frequently exposed to potentially dangerous solutions, it would be nice if the device could perform self diagnostics on the

hardware to ensure that everything is functioning properly. Analogous to PC self diagnostic scans and tests, this would be an ideal feature to include in a commercial portable potentiostat.

In order to make this device more cost competitive and have a technical advantage over current portable potentiostats, it would be ideal if this device had a self calibration feature. This self calibration feature could reference and re-adjust its sensors sensitivity based on known materials, quantities, and signals from an online server. It would be nice if the device could detect the different metals in a solution, and also provide background and information on that particular found metal if user requested it.

Another useful addition would be the ability for the user to adjust the current sensitivity, as this feature is hardwired in the team's final design. Ideally, this would be a feature that would be adjustable via the UI. Adjusting the current sensitivity would allow for higher accuracy on sweeps with small returned currents, and lower accuracy on sweeps with larger returned currents.

# Appendix I

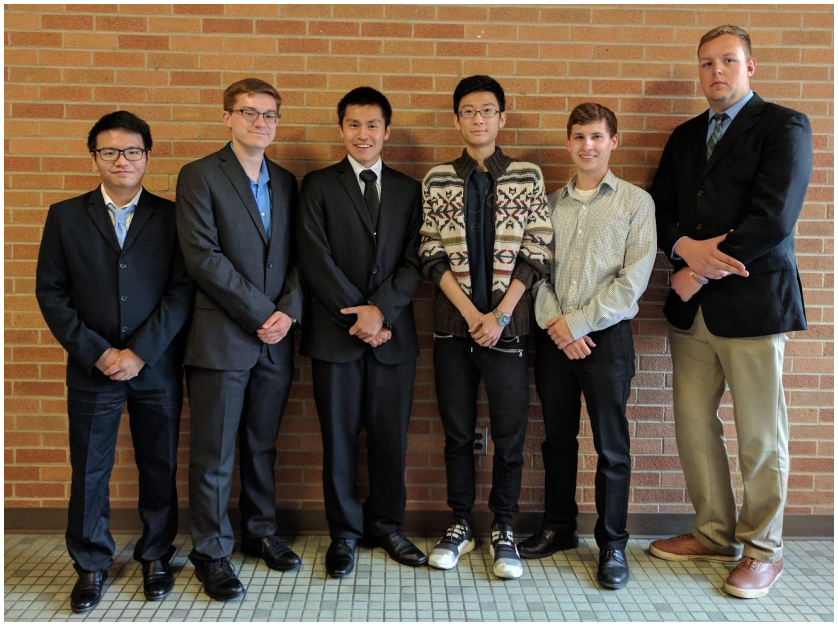

**Figure 61: From Left to Right: Xingchen Xiao, Nathan Bagnall, Haitai Ng, Yuzhou Wu, Justin Opperman, Brandon Gevaert**

## Technical Roles

**Xingchen Xiao : Documentation**

My technical roles for this project is primarily designing and coding the LabVIEW Interface. I also have a team role in documentation. At the beginning of the project, I did research on the software platform for our project. I tried to design and develop an interface by using MFC in C++. However, after the research, I decided to develop the user interface on LabVIEW and change our software design platform. In the first version of the labVIEW, we don't have any connection to the microcontroller so I did a demo waveform with hard-code data. This is for demonstrating the functionalities of the labVIEW user interface. I have design the software function like File management and waveform plotting as our customer required. The software does decent jobs in saving data and reproduce waveforms. In the 2nd and 3rd edition of our LabVIEW project, I made minor change to the path directing of our software to make sure it can create new folder for saving files if the saving path does not exist. And I designed a way to let user to name their saving files for identifying purpose which is adding user-input prefix to the file names. In the documentation part, I participated the presentations, report, and other documentations. I have written most of software-side design and information in these slides.

### Nathan Bagnall: Project Manager

I have had three primary technical roles for this project. First, I was involved in the lengthy decision of choosing a microcontroller to develop our device around. We fluctuated for a while between the Cypress PSoC 4000 and the TI MSP430, but after attempting to develop for both of them we changed directions and focussed on the Arduino Mega with an external Digital-to-Analog converter for a more precise waveform. My biggest technical roles have been designing and building the power supply and designing the enclosure for our final device. For the power supply section we needed a positive and negative rail that were stable and with minimal variation. In order to accomplish this we looked into using a standard “wall-wart” power supply in order to keep it simplistic. Unfortunately, to use these units we would have to compromise on noise output, since almost every power supply in this configuration uses switching regulators and very few offer negative voltages. Another possibility would have been to use a multi-tap isolation transformer, but this would have been heavy and bulky. After debating between these different approaches, I ended up designing one around two 9V batteries and linear regulators to maintain stable voltage levels with minimal noise. This approach gave us both voltage levels, commonly available batteries, and linear regulators giving us high stability. Although 9V batteries only have about 400mAh of capacity, our device draws very little power and therefore the device will be able to run for an extended period of time without changing the batteries. A potential improvement would be to replace the batteries with a rechargeable solution in the future. My final technical contribution was designing the enclosure which was relatively straightforward. I first designed standoffs for the Arduino to mount on, then I designed a box around it and the PCB that would be mounted on top of the Arduino. Finally, I incorporated a sliding mechanism to close the top and make it possible to change out the batteries when they die.

### Haitai Ng : Documentation

My role and contribution to the final design lies in the software for the microcontroller or interface platform. I was responsible for programming the Arduino Mega which would sync the hardware and software into one embedded unit.

In the beginning, our team planned to use the MSP430G2553. During the development I looked into various resources. I completed the Texas Instruments webinar series for the MSP430 family. During the first demo I was able to configure the GPIO pins to accept a voltage, configured the onboard PWM outputs, and plot the voltage in real time using Code Composer Studio’s integrated waveform generator. However after further research I concluded that this particular model of the MSP430 did not have the internal peripherals we would need to complete our project. One major reason why we selected to use this platform was because this model of the MSP430G2553 is widely available in the ECE Capstone lab, as it was used in the previously required ECE 480 laboratory exercises.

I then transitioned and focused on learning how to integrate the Cypress microcontroller. I learned how to use the Cypress Microcontroller by watching the PsoC 6 tutorials taught by Alan Hawse. As stated in our proposal, I found that this device had very

limited documentation, thus making development very difficult. In addition it was difficult to sync with other non-cypress products.

Our third microcontroller selection and the one that was incorporated into our final design was the Arduino Mega. The final embedded software that I developed was completed using C and C++. Using the Arduino IDE, I created the C++ library that would sync exclusively with the Arduino Mega. Throughout the semester I collaborated with Justin and together we developed the linear sweep, pulse sweep, and the Arduino user interface that was used to accept user input commands, and communicate with the LabVIEW user interface. In addition I worked closely with Nathan and we synchronized the external ADC to communicate with our Arduino Mega using I2C (This was later removed). All code can be found on my personal GitHub ( https://github.com/hkng248).

### Yuzhou Wu : Lab Coordinator

My technical roles for this project are parts ordering and doing software data communication between Labview and Arduino. At first, I considered using Microsoft Visual Studio to do the software data communication, but the driver of the Microcontroller and the protocol of serial communication have some problems. So I switched the platform from VS to LabVIEW. LabVIEW has its own driver for different Microcontroller, that means I don't need to spend plenty of time to config the driver. But the problem is the LabVIEW is a piece of software that i've never touched before. So I had to spend time to learn how to use this platform. The project can be roughly divided into three part: hardware circuit, Microcontroller and LabVIEW software. To generate the signal to circuit, the Microcontroller has its own software to read and write. So the LabVIEW part need to integrate with the Microcontroller's software. In the first version of the LabVIEW, it can be connected with Microcontroller, but the problem is the data write and read can't synchronized with Arduino correctly. Sometimes the write loop does not finish, it start to read the data. To avoid this, I started to try using asynchronous process to make read doing after write in the second version of LabVIEW. The second version still has some problems. Because the data transmission is based on serial port, if Microcontroller send too much data too fast through serial port, LabVIEW might lost some data when read from serial port. In the third version, I added a delay in the read loop of LabVIEW, that will give the read function a short delay to avoid data lost. And in the third version, there are some minor changes to increase the efficiency.

### Justin Opperman : Documentation

I worked primarily on the hardware side of the project, and was responsible for building and designing the external circuitry that converts a digital signal from the Arduino into an analog signal at the working electrode. I built the PWM output circuit that was too inaccurate to be used, and then built the DAC circuit as a solution to the accuracy problem. I also built the first current reading prototype circuit we used, and worked with Nathan to design overvoltage protection for it.

I also did work on the embedded software side of the project, writing code that was used to generate an adjustable square wave sweep. I also wrote embedded code that could

sample the measured current at the end of each square wave pulse, and take the difference of the high and low currents. I then worked with Haitai to integrate the square wave code with his code that could output to the external DAC. I discovered why the output of the DAC was unpredictable when changing inputs, and developed the solution to the problem by allowing the write pin on the DAC to be toggled.

In addition, I wrote embedded code that allowed the user to input values and write parameters directly to the Arduino before the LabView UI was integrated. The user could then directly read returned current results that the Arduino was measuring. This allowed the team to test the hardware and embedded software before the Labview UI was fully functional. Haitai made many improvements to my initial testing unit, some of which were integrated into the final design.

I also did lots of testing in order to properly calibrate the potentiostat, which included obtaining test data and creating curve fitting equations that could convert Arduino digital values into the correlating analog values present at the input/output of the potentiostat.

### Brandon Gevaert : Presentation

I primarily worked on the hardware side of the project. This included looking into the Microcontroller selection. While it was temporarily decided to use the MSP430 commonly used in the lab and other projects, I looked into other TI product families that would have allowed for the ability to output files to a bluetooth connected device while only changing the pin layout and basic aspects of the code. I did an initial test that would have allowed the integration to the TI connectivity simplelink system. This was ultimately to help with adding an additional feature to a finalized design, but the switch to the Arduino Mega ultimately changed this to not being useful in the final design. I also was involved in developing the final current sensing portion of the device. This current sensing section included some fairly extensive research into the methods used to sense currents currently used in industrial and research applications. The current sensing circuit was ultimately designed as a result of the research done to find a way to accurately sense high resistance low current outputs. I selected the component values for acceptable tolerances in this circuit to keep the design meeting the standards set by the facilitator and the cost limitations. I simulated the circuit both theoretically using ADS and practically using a breadboard created circuit that would act as a real world test. After this was finalized along with the other aspects of the hardware, I designed the PCB to the specifications set by the PCB manufacturer, the ECE Shop, and based on what they specified as well as our own requirements for space saving designs. The PCB manufacturing also meant importing the designs for the power supply designed by Nathan as well as setting up the circuit in a technical format for the voltage output and current setting sections. This meant connecting together the total system as well as checking the integration for any potential issues that might arise. The PCB component placement was also included in this while autorouting was used to save on time.

# Appendix II

## Course References

1. CSE 231 : Introduction To Programming I
2. CSE 232 : Introduction To Programming II
3. CSE 260 : Discrete Structures In Computer Science
4. CSE 331 : Algorithms and Data Structures
5. CSE 335 : Object Oriented Software Design
6. CSE 410 : Operating Systems
7. ECE 201 : Electric Circuits and Systems I
8. ECE 202 : Electric Circuits and Systems II
9. ECE 203 : Electric Circuits and Systems Laboratory
10. ECE 230 : Digital Logic Fundamentals
11. ECE 280 : Electrical Engineering Analysis
12. ECE 302 : Electronic Circuits
13. ECE 303 : Electronics Laboratory
14. ECE 331 : Microprocessors and Digital Systems
15. ECE 366 : Signal Processing
16. ECE 402 : Application Of Analog Integrated Circuits
17. ECE 404 : Radio Frequency Electronic Circuits
18. ECE 446 : Biomedical Signal Processing

## Textbook References


1. Active and Nonlinear Electronics for Michigan State - Schubert & Kim
2. C++ Primer - Lippman, Lajoie and Moo
3. Chemical Instrumentation: A Systematic Approach Third Edition -
4. Data Structures and Algorithm Analysis in C++ - Weiss, Mark A.
5. Design patterns: elements of reusable object-oriented software - Erich Gamma
6. Digital Design and Computer Architecture ARM Edition - Harris, Sarah
7. Discrete Mathematics and Its Applications - Kenneth Rosen,
8. ECE 201 : Electric Circuits and Systems I - Gregory Wierzba
9. ECE 202 : Electric Circuits and Systems II - Gregory Wierzba
10. ECE 203 : Electric Circuits & Systems Laboratory - Gregory Wierzba
11. ECE 302 : Electronic Circuits - Gregory Wierzba
12. ECE 303 : Electronics Laboratory - Gregory Wierzba
13. ECE 402 : Application Of Analog Integrated Circuits - Gregory Wierzba
14. ECE 404 : Radio Frequency Electronic Circuits - Gregory Wierzba
15. Linear Algebra - Seymour Lipschutz, Marc Lipson

16. Low Level Measurements Handbook 6th edition - Keithley Instruments
17. Making Precision Low Current and High Resistance Measurements - Keithley Instruments
18. Making Sense of Current Sensing (White Paper) - Texas Instruments
19. MICROCONTROLLER THEORY & APPL: HC12 & S12 (W/CD) - PACK
20. Object-oriented modeling and design with UML - Michael Blaha, James Rumbaugh
21. Optimizing Low-Current Measurements and Instruments (White Paper) - Jonathan L. Tucker
22. Practice of Computing Using Python, The (2nd Edition) - William F. Punch, Richard Enbody
23. Solid State Electronic Devices Sixth Edition
24. Square Wave Voltammetry by Osteryoung
25. The Analysis and Design of Linear Circuits - Roland E. Thomas, Albert J. Rosa, Gregory J. Toussaint

## Research Papers

1. A Simplified Microcontroller Based Potentiostat for Low-Resource Applications, Bolaji Aramo et al, Obafemi Awolowo University
2. Building a Microcontroller based potentiostat: A Inexpensive and versatile platform for teaching electrochemistry and instrumentation, Gabriel N. Meloni, Universidade de São Paulo
3. DStat: A Versatile, Open-Source Potentiostat for Electroanalysis and Integration, Michael D. M. Dryden et al, University of Toronto

## Sources

1. http://mspsci.blogspot.com/2010/07/tutorial-01-getting-started.html
2. http://www.cypress.com/blog/psoc-creator-news-and-information/alan-hawse-talks-about-freertos-and-psoc
3. http://www.cypress.com/products/32-bit-arm-cortex-m4-psoc-6
4. http://www.ti.com/lit/ug/slau144j/slau144j.pdf
5. http://www.ti.com/tool/CCSTUDIO
6. http://xanthium.in/ports-of-msp430g2xxx-series
7. http://zone.ni.com/reference/en-XX/help/371361P-01/lvinstio/visa_bytes_at_serial_port/
8. http://zone.ni.com/reference/en-XX/help/371361P-01/lvinstio/visa_read/
9. http://zone.ni.com/reference/en-XX/help/371361P-01/lvinstio/visa_write/
10. http://zone.ni.com/reference/en-XX/help/372614J-01/glang/spreadsheet_str_to_array/
11. https://education.ti.com/en/professional-development/teachers-and-teams/online-learning/live-webinars
12. https://education.ti.com/en/professional-development/teachers-and-teams/online-learning/on-demand-webinars

13. https://en.wikipedia.org/wiki/LabVIEW
14. https://github.com/alanbarr/msp430-launchpad
15. https://github.com/arduino/Arduino
16. https://github.com/WaveShapePlay/ArduinoPySerial_LearningSeries/tree/master/Part6_WritingToTextFile
17. https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf
18. https://www.arduino.cc/
19. https://www.ni.com/visa/
20. https://www.youtube.com/channel/UC8Sp1Q_uTQcvVb0JEgU9jGg
21. https://www.youtube.com/channel/UCOPwgpx4mNwPULlfdCn1xiQ
22. https://www.youtube.com/channel/UCqzyHMbket3E1gS2mAPJ9UA
23. https://e2e.ti.com/blogs_/archives/b/precisionhub/archive/2015/07/10/six-ways-to-sense-current-and-how-to-decide-which-to-use
24. https://training.ti.com/getting-started-current-sense-amplifiers
25. http://www.ti.com/amplifier-circuit/current-sense/overview.html#technotes
26. CheapStat: https://iorodeo.com/products/cheapstat-open-source-potentiostat

# Appendix III

Below are the original and final Gantt charts for this project. If interested, here are links to the original and final Gantt charts.

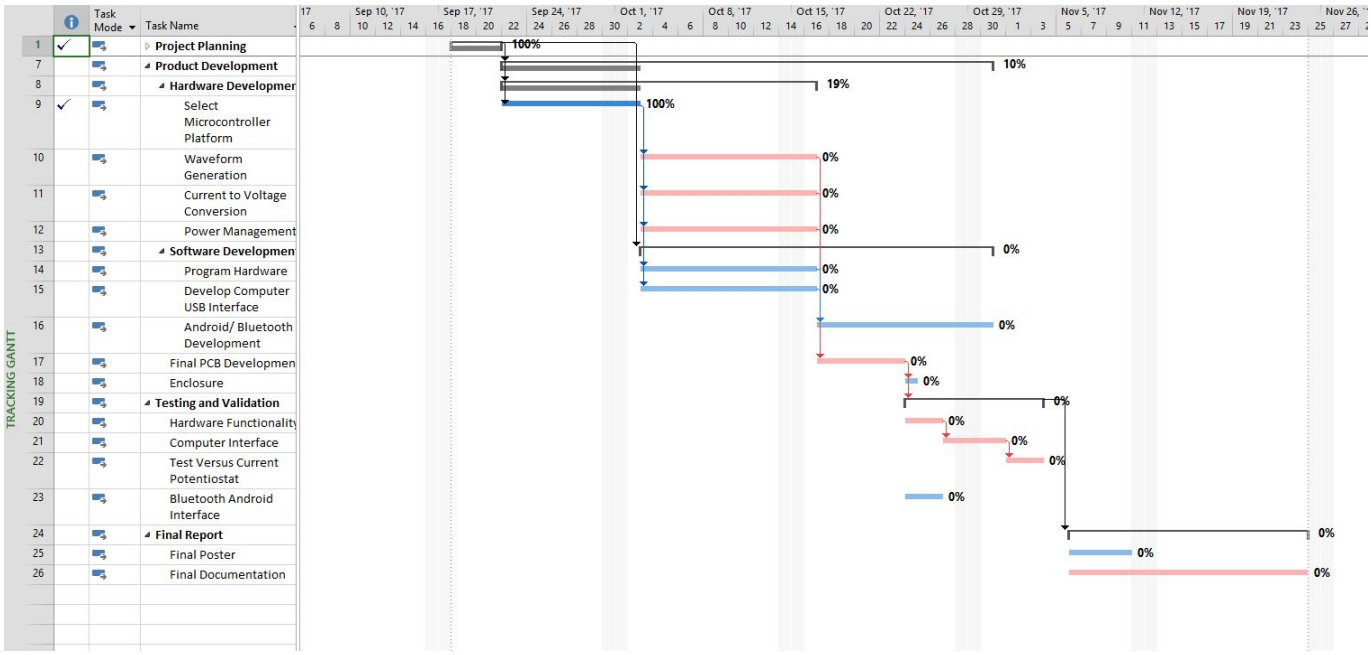


Figure 62: Original Gantt Chart

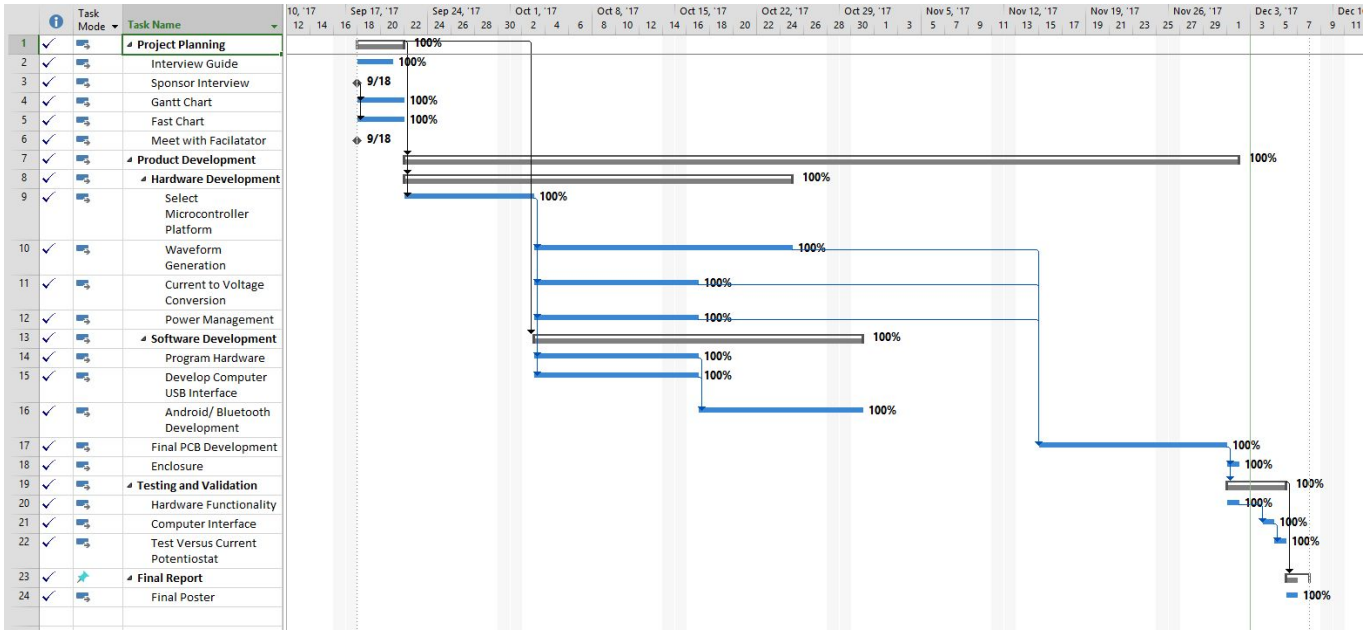


Figure 63: Full size final Gantt Chart

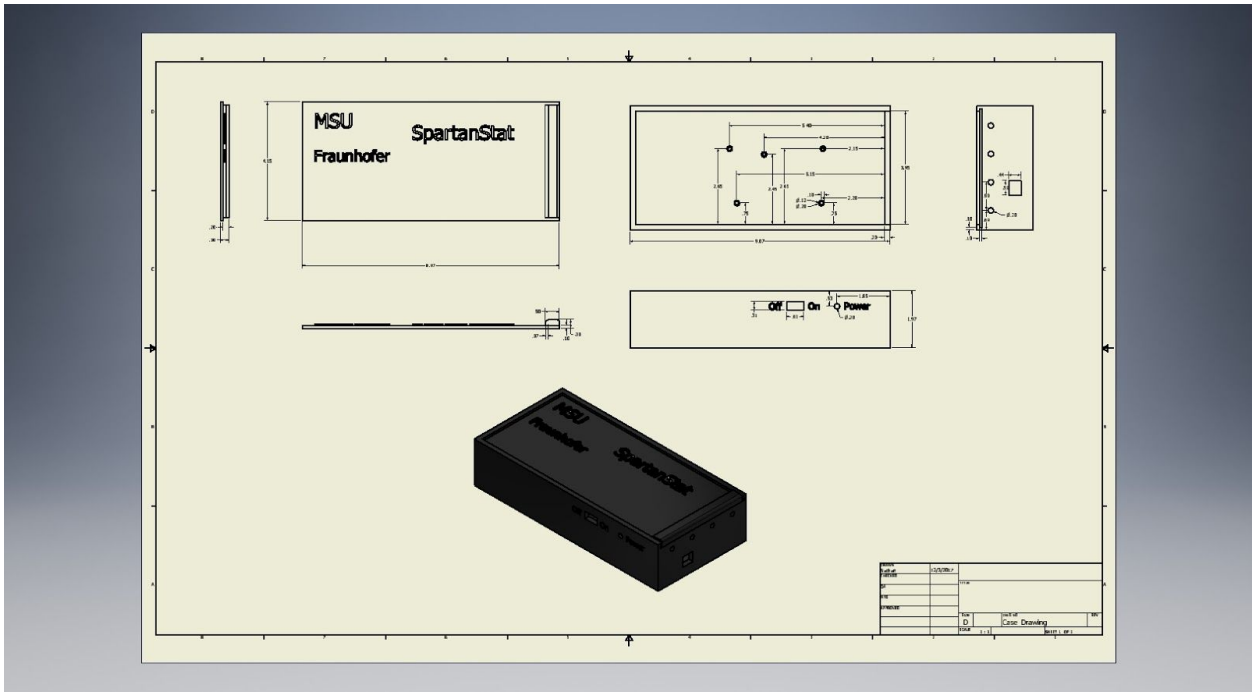


Figure 64: Enclosure CAD Drawing

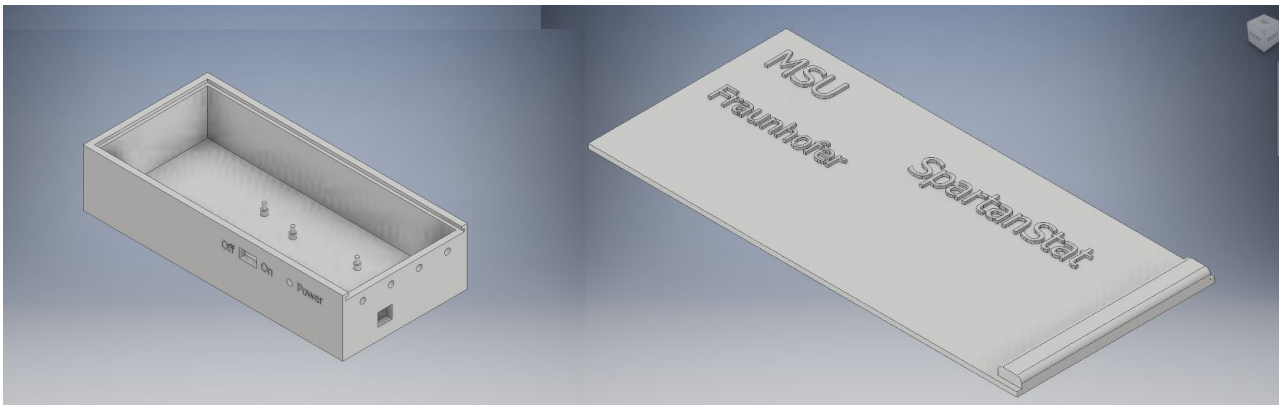


Figure 65: Enclosure Model

# Arduino Code

```
#include "PotentiostatLibrary.h"

float delayTimeHigh;
float delayTimeLow;
float quietTime;
float squareWaveAmplitudeVoltage;
float squareWaveStepVoltage;
float startVoltage;
float stopVoltage;
int linearSweepCount;
int linearSweepAlgorithmSlope;

PotentiostatLibrary potentiostat = PotentiostatLibrary();
void setup()
{
  Serial.begin(9600);
  potentiostat.setPinsToOutput();
  potentiostat.setWritePinToOutput();
}

void loop()
{

  PotentiostatLibrary potentiostat = PotentiostatLibrary();

  Serial.println("---------- Program Executed ------------");
  Serial.println("Please enter P for pulse // L for linear // C for current read");
  while(Serial.available() == 0){}


  if(Serial.available())
  {
    String userInput = Serial.readString(); // Serial read converts characters to ASCII
equivalent
    int operationCompleted = 0;

    if(userInput == "P") // If user equals pulse: "P" execute pulse wave
    {
      operationCompleted = executeSquareWave();
      Serial.print("Completed Operation Return Value: ");
```

```
  Serial.println(); Serial.println();
}
else if (userInput == "L") // If user enters linear: "L" execute linear wave
{
 Serial.println("Executing linear sweep");
 Serial.println("Enter number of repititions");
 while(Serial.available() == 0) {} // wait for user inpiut
 linearSweepCount = Serial.parseInt();
 potentiostat.setLinearSweepRepititions(linearSweepCount);

 Serial.println("Enter Start Voltage (mV):");
 while(Serial.available() == 0) {}
 startVoltage = Serial.parseFloat();
 Serial.println("Enter Stop Voltage (mV):");
 while(Serial.available() == 0) {}
 stopVoltage = Serial.parseFloat();
 potentiostat.setLinearParameters(-startVoltage, -stopVoltage);
 Serial.println("Please indicate whether linear sweep slope is increasing to decreasing");
 Serial.println("-- Options: decreasing to increasing 1 // increasing to decreasing : 2 -- ");
 Serial.println("Enter slope (1 or 2): ");
 while(Serial.available() == 0) {}
 linearSweepAlgorithmSlope = Serial.parseInt();
 potentiostat.setLinearSlope(linearSweepAlgorithmSlope);
 displayLinearSweepParametersToConsole();

 potentiostat.executeLinearSweep();
 Serial.print("Completed Operation Return Value: ");
 Serial.println(); Serial.println();
}
else if (userInput == "C")
{
  int x = 1;

  Serial.println("Executing Read Current");
     while(x > 0)
     {
       operationCompleted = potentiostat.readCurrent();
     }
  Serial.print("Completed Operation Return Value: ");
  Serial.print(operationCompleted);
  Serial.println(); Serial.println();
```

```
    }
    else
    {
      Serial.println("Error :: Invalid Input. Please try again \n");
    }
  }
}

int executeSquareWave()
{
  Serial.println("Enter Square Wave Parameters: ");
  Serial.println("Enter Quiet Time (ms)");
  while(Serial.available() == 0){} // Wait for user input
  quietTime = Serial.parseFloat();
  Serial.println("Enter Delay Time High (ms)");
  while(Serial.available() == 0){} // Wait for user input
  delayTimeHigh = Serial.parseFloat();
  Serial.println("Enter Delay Time Low (ms)");
  while(Serial.available() == 0){} // Wait for user input
  delayTimeLow = Serial.parseFloat();
  Serial.println("Enter Square Wave Amplitude Voltage (mV)");
  while(Serial.available() == 0){} // Wait for user input
  squareWaveAmplitudeVoltage = Serial.parseFloat();
  Serial.println("Enter Square Wave Step Voltage (mV)");
  while(Serial.available() == 0){} // Wait for user input
  squareWaveStepVoltage = Serial.parseFloat();
  Serial.println("Enter Start Voltage (mV)");
  while(Serial.available() == 0){} // Wait for user input
  startVoltage = Serial.parseFloat();
  Serial.println("Enter Stop Voltage (mV)");
  while(Serial.available() == 0){} // Wait for user input
  stopVoltage = Serial.parseFloat();
  displaySquareWaveParametersToConsole();

  potentiostat.init(delayTimeHigh, delayTimeLow, quietTime,
squareWaveAmplitudeVoltage, squareWaveStepVoltage, startVoltage, stopVoltage);
  Serial.println();
  return 1;
}

void displaySquareWaveParametersToConsole()
{
```

```
  Serial.println();
  Serial.print("Quiet Time (ms): ");
  Serial.println(quietTime);
  Serial.print("Delay Time High (ms): ");
  Serial.println(delayTimeHigh);
  Serial.print("Delay Time Low (ms): ");
  Serial.println(delayTimeLow);
  Serial.print("Square Wave Amplitude Voltage (mV): ");
  Serial.println(squareWaveAmplitudeVoltage);
  Serial.print("Square Wave Step Voltage (mV): ");
  Serial.println(squareWaveStepVoltage);
  Serial.print("Start Voltage (mV): ");
  Serial.println(startVoltage);
  Serial.print("Stop Voltage (mV): ");
  Serial.println(stopVoltage);
}

void displayLinearSweepParametersToConsole()
{
  Serial.println();
  Serial.print("Number of repititions: ");
  Serial.println(linearSweepCount);
  Serial.print("Stop Voltage (mV): ");
  Serial.println(startVoltage);
  Serial.print("Start Voltage (mV): ");
  Serial.println(stopVoltage);
  Serial.println();
}
```

**Figure 66 : Potentiostat Complete Module (Arduino File)**

```
/**
 * \file PotentiostatLibrary.h
 *
 * \author Haitai Ng & Justin Opperman
 *
 * This file contains all the function declarations and member variables
 * required to execute the portable potentiostat created by MSU
 * ECE 480 Team 11 Fall 2017.
 *
```

```
 * This file is to be used with PotentiostatLibrary.cpp
 */


#ifndef PotentiostatLibrary_h // TL
#define PotentiostatLibrary_h // TL

#if (ARDUINO >= 100)
  #include "Arduino.h"
  #include "Adafruit_ADS1015.h"
  #include "Wire.h"
#else
  #include "WProgram.h"
  #include "pins_arduino.h"
  #include "WConstants.h"
#endif


/**
 * Potentiostat Library is an object that is to be used with the developed
 * PCB / Circuit made by the ECE 480 Team 11.
 */
class PotentiostatLibrary
{
  public:

    /// Constructor
    PotentiostatLibrary();
    /// Initialize and pass square wave parameters to private member variables
    void init( double delayTimeHigh, double delayTimeLow,
      double quietTime, double squareWaveAmpltudeVoltage,
      double squareWaveStepVoltage, double startVoltage, double stopVoltage);

    /// Set GPIO pins to output
    void setPinsToOutput();
    /// Set GPIO pins to input
    void setPinsToInput();
    /// Set 'Write Pin' to output a digital high signal
    void setWritePinToOutput();
    /// Write value to digital GPIO pins (logic low or logic high)
    void writeToDigitalPin( int digitalPin, int value);
    /// Output voltage from GPIO pin
```

```
    void outputDigitalValues( int digitalEquivalentValue);
    /// Calibrate a desired voltage to be compatible with the external DAC
    int convertVoltageForDAC(int desiredVoltage);
    /// Read current
    int readCurrent();
    /// Convert a digital signal from the DAC to a voltage
    double DACvalToVoltage(int DACval);


    /// Linear sweep functions
    void linearSweepAlgorithm();
    /// Set member variables relatives to linear sweep
    void setLinearParameters(double startVoltage, double stopVoltage) {mStartVoltage =
startVoltage; mStopVoltage = stopVoltage;}
    /// Set the slope of the initial linear sweep
    void setLinearSlope(int slope) {mLinearSweepType = slope;}
    /// Set the number of repitions for linear sweep
    void setLinearSweepRepititions(double repititions) {mLinearSweepRepititions =
repititions;}
    /// Execute the linear sweep
    int executeLinearSweep();

    /// Square wave functions
    void squareWaveAlgorithm( double delayTimeHigh, double delayTimeLow,
      double startValue, double stopValue, double squareWaveAmplitude,
      double squareWaveStep);
    /// Execute the square wave
    void executePulse();
    /// Read the current from the square wave
    double readCurrentForSquareWave();
    /// Print the current onto the console
    void printCurrentForSquareWave(double current);

    /// Pin declarations. These pins output a voltage to the external DAC
    int mDigitalPinZero = 24;
    int mDigitalPinOne = 25;
    int mDigitalPinTwo = 26;
    int mDigitalPinThree = 27;
    int mDigitalPinFour = 28;
    int mDigitalPinFive = 29;
    int mDigitalPinSix = 30;
    int mDigitalPinSeven = 31;
```

```
    int mDigitalPinEight = 32;
    int mDigitalPinNine = 33;
    int mDigitalPinTen = 34;
    int mDigitalPinEleven = 35;

    // Output pin to control writing to the DAC
    int mDigitalPinWR = 23;

    // Square wave parameters
    int mQuietTime = 0;
    int mDelayTimeHigh = 0;
    int mDelayTimeLow = 0;
    double mSquareWaveAmplitudeVoltage = 0;
    double mSquareWaveStepVoltage = 0;
    double mStartVoltage = 0;
    double mStopVoltage = 0;

    /// Linear Sweep
    double mLinearSweepRepititions = 0;
    double mLinearSweepMinVoltage = -2.0;
    double mLinearSweepMaxVoltage = 2.0;
    double mLinearSweepStep = 0.001;
    int mLinearSweepType = 0;

    /// Arduino Mega Pins
    int mAnalogPinOne = A15;
    int mAnalogPinTwo = A14;
    double mADCValueToVoltageRatio = 4878.0 / 1024.0;
    double mADCValue = 0;
    double mCurrent = 0;
};
#endif
```

Figure 67 : PotentiostatLibrary.h

```
/**
 * \file PotentiostatLibrary.cpp
 *
 * \author Haitai Ng & Justin Opperman
 *
```

```
 * This file contains all the function definitions and source code
 * required to execute the portable potentiostat created by MSU
 * ECE 480 Team 11 Fall 2017.
 *
 * This file is to be used with PotentiostatLibrary.h
 */


#include "PotentiostatLibrary.h"
#include "Arduino.h"

Adafruit_ADS1115 ads;

/*
 * Name: PotententioStatLibrary
 * Description: Constructor. Initiate and create a Potentiostat object
 */
PotentiostatLibrary::PotentiostatLibrary()
{
  /// Set the gain and configuration for the external ADC
  ads.setGain(GAIN_TWOTHIRDS);
  ads.begin();
}

/*
 * Name: convertVoltageForDAC
 * Description: Calbirate the desired voltage so it can be received by DAC
 */
int PotentiostatLibrary::convertVoltageForDAC(int desiredVoltage)
{
  return (desiredVoltage + 2346) * 0.681;
}

/*
*  Name: init
*  Description: initialize and set private member variables associated with
*   the square wave
*
* \param Square Wave input parameters
* \return None
*/
```

```
void PotentiostatLibrary::init( double delayTimeHigh,
  double delayTimeLow, double quietTime, double squareWaveAmpltudeVoltage,
  double squareWaveStepVoltage, double startVoltage, double stopVoltage)
{
  mDelayTimeHigh = delayTimeHigh;
  mDelayTimeLow = delayTimeLow;
  mQuietTime = quietTime;
  mSquareWaveAmplitudeVoltage = 0.681 * squareWaveAmpltudeVoltage;
  mSquareWaveStepVoltage = 0.681 * squareWaveStepVoltage;
  mStartVoltage = convertVoltageForDAC(startVoltage);
  mStopVoltage = convertVoltageForDAC(stopVoltage);
  executePulse();
}

/*
*  Name: setPinToOutput
*  Descripton: set the GPIO pins to output mode
* \param None
* \return None
*/

void PotentiostatLibrary::setPinsToOutput()
{
  pinMode(mDigitalPinEleven, OUTPUT);
  pinMode(mDigitalPinTen, OUTPUT);
  pinMode(mDigitalPinNine, OUTPUT);
  pinMode(mDigitalPinEight, OUTPUT);
  pinMode(mDigitalPinSeven, OUTPUT);
  pinMode(mDigitalPinSix, OUTPUT);
  pinMode(mDigitalPinFive, OUTPUT);
  pinMode(mDigitalPinFour, OUTPUT);
  pinMode(mDigitalPinThree, OUTPUT);
  pinMode(mDigitalPinTwo, OUTPUT);
  pinMode(mDigitalPinOne, OUTPUT);
  pinMode(mDigitalPinZero, OUTPUT);
}

/*
*  Name: setWritePinToOutput
*  Description: set the write pin to output mode
* \param None
* \return None
```

```
*/
void PotentiostatLibrary::setWritePinToOutput()
{
  pinMode(mDigitalPinWR, OUTPUT);
}

/*
*  Name: WriteToDigitalPins
*  Description: Set the value to the GPIO pin.
*   Logic high = a voltage > 0
    Logic low = a voltage < 0
* \param integer GPIO pin , integer Logic high (1) or Logic low (0)
* \return None
*/
void PotentiostatLibrary::writeToDigitalPin( int digitalPin, int value)
{
  if(value == 1)
  {
    digitalWrite(digitalPin, HIGH);
  //  Serial.print("1");

  }
  else
  {
    digitalWrite(digitalPin, LOW);
  //  Serial.print("0");
  }
}

/*
*  Name: OutputDigitalPins
*  Description: Convert an integer value into a 12 bit binary equivalent
*  Creates an array data structure (size 12) and converts a 32 bit integer value
*  into a (12 bit) binary equivalent
*  This is completed by extensive bit shifting.
*  Each bit of the 12 bit array will then be outputted to the DAC by writing
*  Initialize and write once the digital write pin has been set
*
*
* \param integer DigitalEquivalentValue
* \return None
*/
```

```
void PotentiostatLibrary::outputDigitalValues( int digitalEquivalentValue)
{
 int a = 31; // 32 bit number
 int referenceIndex = 0;
 int splitArray[12]; // integers are 32 bit, create a 12 bit array
 while( a > 0)
 {
  int digitalOutputPinArray[32]; // initalize the array
  int n = digitalEquivalentValue >> a; // bit shifting to signed
  if (!(n % 2)) digitalOutputPinArray[referenceIndex] = 0 ;
  else digitalOutputPinArray[referenceIndex] = 1;
  a--;
  referenceIndex++;
  //copy over the 12 bits of interest into the smaller array
  memcpy(splitArray, digitalOutputPinArray + 20, sizeof(splitArray));
 }

 /// set the GPIO pins to output its associated value
 writeToDigitalPin(mDigitalPinEleven, splitArray[0]); // 2^11
 writeToDigitalPin(mDigitalPinTen, splitArray[1]); // 2^10
 writeToDigitalPin(mDigitalPinNine, splitArray[2]); // 2^9
 writeToDigitalPin(mDigitalPinEight, splitArray[3]); // 2^8
 writeToDigitalPin(mDigitalPinSeven, splitArray[4]); //2^7
 writeToDigitalPin(mDigitalPinSix, splitArray[5]); // 2^6
 writeToDigitalPin(mDigitalPinFive, splitArray[6]); // 2^5
 writeToDigitalPin(mDigitalPinFour, splitArray[7]); // 2^4
 writeToDigitalPin(mDigitalPinThree, splitArray[8]); // 2^3
 writeToDigitalPin(mDigitalPinTwo, splitArray[9]); // 2^2
 writeToDigitalPin(mDigitalPinOne, splitArray[10]); // 2^1
 writeToDigitalPin(mDigitalPinZero, splitArray[11]); // 2^0

 delay(1);
 digitalWrite(mDigitalPinWR, LOW); //< Disable all writing to DAC
 delay(1); //< wait until all GPIO pins have all their values set
 digitalWrite(mDigitalPinWR, HIGH); //< write to DAC
 delay(1);

}

/*
*  Name: linearSweepAlgorithm
```

```
*  Description: This program executes the linear sweep
* \param None
* \return None
*/
void PotentiostatLibrary::linearSweepAlgorithm()
{
  double delayTimeHigh = 0; double delayTimeLow = 0;
  double startValue = 0; double stopValue = 0;
  double squareWaveAmplitude = 1;
  double squareWaveStep = 1;


  /// If linearSweepType == 2
  /// The initial slope is positive and Increasing
  if(mLinearSweepType == 2)
  {
    startValue = convertVoltageForDAC(mStartVoltage);
    stopValue = convertVoltageForDAC(mStopVoltage);
    // Increasing slope
    for(int val = startValue ; val >= stopValue; val -= squareWaveStep)
     {
      Serial.print(DACvalToVoltage(val), 4);
      Serial.print("     "); // this has to be 5 spaces
      Serial.print("     ");
      outputDigitalValues((int) val);
      readCurrent();
    }

    // Decreasing slope
    for(int val = stopValue; val <= startValue; val += squareWaveStep)
    {
      Serial.print(DACvalToVoltage(val), 4);
      Serial.print("     ");
      Serial.print("     ");
      outputDigitalValues((int) val);
      readCurrent();
    }
  }
  /// If linearSweepType == 1
  /// The initial slope is negative and decreasing
  else if (mLinearSweepType == 1)
  {
```

```
    startValue = convertVoltageForDAC(mStopVoltage);
    stopValue = convertVoltageForDAC(mStartVoltage);

    // decreasing slope
    for(int val = stopValue; val <= startValue; val += squareWaveStep)
    {
      Serial.print(DACvalToVoltage(val), 4);
      Serial.print("     ");
      Serial.print("     ");
      outputDigitalValues((int) val);
      readCurrent();
    }

    /// increasing slope
    for(int val = startValue ; val >= stopValue; val -= squareWaveStep)
     {
      Serial.print(DACvalToVoltage(val), 4);
      Serial.print("     "); // this has to be 5 spaces
      Serial.print("     ");
      outputDigitalValues((int) val);
      readCurrent();
    }
   }
   else
   {
     Serial.println("Error :: Invalid input. Please try again \n");
   }
}

/*
*  Name: executeLinearSweep
*  Description: Run the linear sweep
* \param None
* \return an integer indicating that progress has been completed
*/
int PotentiostatLibrary::executeLinearSweep()
{
  while(mLinearSweepRepititions > 0)
  {
    linearSweepAlgorithm();
    mLinearSweepRepititions--;
  }
```

```
  return 1;
}


/*
*  Name: executePulse
*  Description: initalize and execute the square wave voltammetry
* \param None
* \return None
*/
void PotentiostatLibrary::executePulse()
{
  delay(mQuietTime);
  while(true)
    {
      /// Execute the square wave algorithm
    squareWaveAlgorithm( mDelayTimeHigh, mDelayTimeLow, mStartVoltage,
    mStopVoltage, mSquareWaveAmplitudeVoltage, mSquareWaveStepVoltage);
    delay(5000);
    break;
    }

}

/*
*  Name: squareWaveAlgorithm
* \param square wave parameters
* \return None
*/
void PotentiostatLibrary::squareWaveAlgorithm(double delayTimeHigh, double
delayTimeLow,
  double startValue, double stopValue, double squareWaveAmplitude, double
squareWaveStep)
{
   /*
   * This function creates the increasing part of the square wave. Each iteration of the for
loop generates a pulse with an amplitude specified by the user.
   * After each iteration, the step is incremented, and the next pulse will start at that step.
   */
  for(int val = startValue ; val <= (stopValue - squareWaveAmplitude); val +=
squareWaveStep)
   {
```

```
    // give low part of square wave
    outputDigitalValues((int) val);
    delay(delayTimeLow);
    // ** Here we collect sample current
    double current_high = readCurrentForSquareWave();

    // give high part of square wave
    val += squareWaveAmplitude;
    outputDigitalValues((int) val);
    delay(delayTimeHigh);
    //  ** Here we collect sample current
    double current_low = readCurrentForSquareWave();
    // Reset squarewave back to low, let the loop increment
    val -= squareWaveAmplitude;
    // Print difference of two currents and the starting Voltage
    Serial.print(DACvalToVoltage(val));
    Serial.print("         ");
    double differenceCurrent = current_high - current_low;
    printCurrentForSquareWave(differenceCurrent);
   }
}

/*
*  Name: readCurrent
* \param None
* \return current
*/
int PotentiostatLibrary::readCurrent()
{
  /*
  int16_t adc1;
  adc1 = ads.readADC_SingleEnded(1) * 0.1875; //* 0.1875;
  Serial.print(adc1);
  Serial.println("  ");
  return adc1;
 */
  double ADCValue = analogRead(mAnalogPinOne) * mADCValueToVoltageRatio;
  double current = .0372 * ADCValue - 86.031;
  return current;
```

```
}

/*
 * Name: readCurrentForSquareWave
 * Description: This function reads the raw ADC value from the Arduino pin, and converts
 * it into a current using a calibration equation.
 * \param None
 * \return double Current from square wave
*/
double PotentiostatLibrary::readCurrentForSquareWave()
{
  double ADCValue = analogRead(mAnalogPinOne) * mADCValueToVoltageRatio;
  double current = .0372 * ADCValue - 86.031;
  return current;
}

/*
 * Name: printCurrentForSquareWave
 * Description: This function prints the current in the correct format
 * \param double current
 * \return None
*/
void PotentiostatLibrary::printCurrentForSquareWave(double current)
{
    Serial.print(current);
    Serial.println("  ");
}

/*
* Name: DACvalToVoltage
* Description: Takes a DAC value as input, and returns the corresponding voltage that
should
    appear at the output of the circuit
* \param DACValue
* \return voltage
*/
double PotentiostatLibrary::DACvalToVoltage(int DACval)
{
  double convertedVoltage = (DACval * 1.468) - 2346;
  return -convertedVoltage;
}
```

Figure 68 : PotentiostatLibrary.cpp